PC-9800 series

EGCによる高速化対応

GDCテクニカルブック

GRAPHIC DISPLAY CONTROLLER TECHNICAL BOOK

星野進二＝著

SOFT BANK
ソフトバンク

PC-9800 series

EGCによる高速化対応

GDCテクニカルブック

GRAPHIC DISPLAY CONTROLLER TECHNICAL BOOK

星野進二＝著

ソフトバンク

PC-9800 series

EGCによる高速化対応

GDCテクニカルブック

GRAPHIC DISPLAY CONTROLLER TECHNICAL BOOK

星野進二＝著

ソフトバンク

はじめに

GDCが発表されてから，はやくも8年がたとうとしています。GDCが最初に搭載されたコンピュータは，NECのN 5200だったと記憶しています。が，広く知られるようになったのは，NECのベストセラーパソコンPC-9801に搭載されてからでしょう。GDCの描画速度のお陰でPC-9801は，従来のパーソナルコンピュータとは一線を画する描画速度をもったマシンとなりました。そのためCADなど従来パソコンでは考えられなかった用途にも，PC-9801が使われるようになりました。また，ゲームやグラフィックスなどで数100本の直線によるワイヤーフレームモデルが，リアルタイムで動くようになったのもPC-9801＋GDCのお陰だと思います。

しかし，このGDCは高速な描画速度をもっている半面，プログラムを作るうえでは，なかなかたいへんなものです。その理由は，GDCへのコマンドおよびパラメータがとても複雑だからです。かなり低レベルなパラメータまで計算しなければなりません。たとえば直線を引く場合，始点・終点の座標さえ与えればよさそうなものですが，実際には描画開始アドレス，ドットアドレス，描画方向，描画パラメータなどを計算しなければなりません。

また，PC-9801でGDCのプログラムを作るには，PC-9801特有なことも知っておかなければなりません。そこで本書では，PC-9801上のGDC描画プログラムにテーマをしぼって解説しています。また，最近のPC-9801には，EGC(Enhanced Graphic Charger)が搭載されていますが，このEGCを使った高速化についても「第4章　高速化」で多少触れています。

本書のプログラムは，大半がLattice C Compilerのソースプログラムで書かれています。現在では，わかりやすさと普及度の面でCが最適だと考えたからです。実用プログラムを作るにはこれらをアセンブリ言語にする必要があります。そのために後半部では，Cプログラムをアセンブリ言語プログラムに書き直しています。

本書に掲載されているプログラムが，みなさんの作成するGDCプログラムの参考になれば幸いです。

最後に，本書がこのように刊行されるのは，遅い筆に叱咤激励してくださった日本ソフトバンク　ハードウェア活用書編集部の麻生健司氏のおかげです。また，たいへんご迷惑をお掛けした棚木光代氏をはじめ日本ソフトバンクの方々に，この場を借りてお礼を申し上げます。

1989年10月　　　　　　　　　　　　　　　　　　　　　　著者

目次

第3章 GDCの描画例 85

目次

第4章 高速化 263

第1章

GDCとは

1 GDCの特徴

GDC(Graphic Display Controller)は，グラフィックスLSIの一種で，ディスプレイ(CRT)に図形や文字を表示するためのものです。GDCの特徴は描画機能を有することです。発表は昭和56年だったと記憶していますが，当時この描画機能は画期的なものでした。直線や円を，単に命令を与えるだけで非常に高速で描画してくれるのですから。GDCを搭載していないパソコンでも直線や円は描画できましたが，CPUでアドレスを計算しながら1ドットずつ点を描画していました。これでは，とても遅い描画しか行えません。GDCは1ピクセル800nsで描画することができました(μPD7220)。直線や円を目にも止まらぬ速さで描画できます。このお陰で，GDCを搭載したパソコンでも，100本程度の直線で構成されたワイヤーフレームの3次元グラフィックスが，リアルタイムで動くようになりました。また，CAD(Computer Aided Design)など非常にたくさんの直線を描画する必要のあるアプリケーションソフトも，パソコン上でなんとか動作するようになりました。PC-9801で高速なグラフィックスソフトが多数動作するのも，内蔵しているGDCに負うところが大きいといえます。

描画機能のほかにもGDCは，表示領域をスクロールさせる機能や画面を横に分割する機能など，いろいろ魅力的な機能をもっています。本書では，特にGDCの描画機能を使ったプログラミングについて解説をしていきます。この章では，GDCの特徴を眺めていきましょう。

1 高速グラフィック描画

GDCのもっとも重要な特徴に，高速グラフィック描画機能があります。GDCには

- μPD7220A
- μPD7220A-1
- μPD7220A-2

の3種類があり，それぞれの最大動作クロックは6MHz，7MHz，8MHzです。最

大動作クロックで動作させた場合，1ドットを描画するために要する時間は，それぞれ660ns，580ns，500nsです。PC-9801に搭載されているGDCは2.5MHzと5MHzで動作しますが，5MHz動作時には約800ns/ドットということになります。800ns/ドットの場合，画面の右上から左下に直線を引くと

800(ns/ドット)×640（ドット）＝512(μs)

かかりますので

1(s)÷512(μs)＝1953

により，1秒間に1953本の直線を引くことができる計算になります。実際には，描画を開始するまでの処理時間がかなり入ってくるので，ここまでの速度は出ませんが，非常に速いことがわかると思います。

GDCは，直線のほかに円/円弧/四辺形/四辺形塗りつぶし/点/グラフィックス文字などを高速に描画することができます。いったん描画を開始してしまえば非常に高速ですが，描画する前にはさまざまなパラメータを計算したり，コマンド/パラメータをCPUからGDCに送出したりしなければなりません。また，送出されたコマンド/パラメータをGDCが解釈するのにも時間がかかります。GDCを使って高速なプログラムを書くときには，パラメータの計算および送出に要する時間を最大限短くなるようにすることが重要です。

■主記憶とは独立した大容量メモリ直接制御

GDCには，アドレス線が18本ありますので

2^{18}÷1024＝256

により，最大256Kワード(512Kバイト)のメモリを管理することができます。GDCのアドレスはワード単位にふられています。1アドレス16ビットなので，最大2048×2048ドットのVRAMを構成することができます。PC-9801は96Kバイトのグラフィック VRAMをもっていますが，GDCのメモリ管理能力からいえば，まだまだ余裕があるといえます。

GDCにDRAM(Dynamic RAM)を接続した場合，GDCはDRAMのリフレッシュ動作を行ってくれます。また，主記憶と独立したところにVRAMをもってくることができるので，CPUとメモリの競合を避けたシステムを容易に構築することができます。ただし，パソコンの場合，VRAMをCPUが直接アクセスできないと，画像データのセーブ/ロードなどにとても長い時間がかかってしまい，具合が悪いも

のです。これはVRAMのデータをGDCが読み，FIFO経由でCPUが読み出すという二度手間のためです。PC-9801では，VRAMがGDCからもCPUからも読み書きできるハードウェア構成になっています。そのためCPUによるVRAMの読み出し/書き込みは非常に高速です。パソコンの場合，VRAMがCPUのメモリ空間にないと使いづらいマシンになってしまうようです。逆に，VRAMがCPUのメモリ空間にあると，CPUとGDCのメモリの競合が生じる欠点もあり，CPUで読み書きするときには，GDCが描画中でないかどうかを確認する必要があります。GDCはこれらのメモリをVRAMとして使用していますが，GDCを単に大容量のメモリ管理のためだけに利用する使い方もあり得ます。

2 16×9ビットの入出力FIFO内蔵

GDCはCPUからコマンド/パラメータを送出されることで動作します。もし，ひとつのコマンド/パラメータしか受け付けないと，CPUはGDCがコマンド/パラメータを解釈する間，待っていなければならないことになります。そんな無駄を省くためGDCは，16個のFIFO(First In First Out)を内蔵しています。このFIFOのおかげで，CPUはコマンドやパラメータを最大16個まで連続してGDCに送出できます。FIFOをうまく活用すれば，GDCに描画コマンドを送出し，GDCが描画している間に，CPUではほかの処理を行うといった効率的なプログラミングが可能になります。

FIFOは逆にGDCからCPUにデータを渡すときにも使います。GDCからCPUに読み出すデータとしては，「VRAMのデータ」や「ライトペンの座標」，「直前の描画が終了した座標」などがあります。FIFOは入力と出力を共用しています。したがって，VRAMデータをFIFOから読み出していても，コマンドやパラメータを送出するとFIFO内のVRAMデータは消えてしまいます。

FIFOはCPUのI/O空間に割り当てられます。PC-9801では

- ●コマンド出力ポート…………A2H
- ●パラメータ出力ポート………A0H
- ●データリードポート…………A2H

となります。A2H，A0HにコマンドやパラメータをOUTすることでFIFOにデータが積まれます。また，A2HからINすることでFIFOの内容がひとつずつ転送

されます。先ほど，GDC経由でVRAMのデータを読み出すためには時間がかかると述べましたが，それはこのFIFO経由でデータを1バイトずつ読み出すことになるからです。

3 メモリ-メモリ間DMA転送機能

GDCはVRAMとCPUのメモリ間をDMA転送することができます。DMAといえば，通常連続したメモリブロックの転送をさしますが，GDCはそのような1次元的なDMAに加えて，2次元的な矩形領域のDMA転送も行うことができます。2次元DMAの場合，矩形領域を45°傾けることも可能です。ただし，GDCのDMAは，通常のDMAに比べるとかなり遅いようです。

PC-9801のハードウェア構成では，GDCのDMA機能は使えません。

4 外部同期機能

GDCはマスターモード/スレーブモードをもっています。スレーブモード時には，外部からの同期信号を受け付け，内部の同期信号発生器を初期化できます。たとえば，外部のビデオ信号と同期をとり，スーパーインポーズすることもできるわけです。また，複数のGDCを接続し，ひとつをマスター，ほかをスレーブ動作させることもできます。

PC-9801はGDCをふたつ搭載しています。ひとつはテキスト制御用に使い，もうひとつはグラフィック制御用に使っています。テキスト制御用GDCがマスター動作を，グラフィック制御用GDCがスレーブ動作をしています。

5 描画タイミングの選択機能

表示期間中にGDCで描画を行うと，表示部と描画部でVRAMデータの競合が起こり，画面が乱れてしまいます。これを**フラッシュモード**と呼びます。一方，表示期間中に描画しないと画面は乱れませんが，描画できる時間が少なくなり遅い描画になってしまいます。これを**フラッシュレスモード**と呼びます。フラッシュモードのほうがフラッシュレスモードの約5倍高速です。GDCは，いずれの描画も行え

るようになっています。描画時間が特に問題になる場合はフラッシュモードを使い，画面品質が重要な場合はフラッシュレスモードを使います。

しかし，最近のPC-9801には，VRAMにデュアルポートメモリが搭載されているため，フラッシュモードでも画面がちらつきません。つまり，フラッシュレスモードの画面品質とフラッシュモードの描画速度が達成されているわけです。

6 表示領域と映像メモリの水平方向の大きさ定義

GDCはディスプレイに表示する画面の大きさを設定できます。通常，画面とVRAMは同じ大きさで使いますが，画面よりも大きな領域を指定することもできます。つまり，GDCでは画面の設定とVRAMの水平方向の大きさを独立に設定できるわけです。表示領域よりも大きなVRAMを設定することで，表示位置を自由に，そして瞬時に切り替えることができます。

PC-9801の場合，640×400モード時には画面とVRAMが同じ大きさになっています。このままでは画面を切り替えて表示することはできません。その代わりVRAMを2組もっており，それらを切り替えて表示したり描画したりすることができるようになっています。

7 拡大表示/拡大描画，パニング，スクロール

GDCはVRAMのデータを拡大して表示する機能をもっています。これを**拡大表示機能**と呼びます。拡大率は1～16倍までを選択できます。この機能は，GDCだけでは行えません。表示アドレスの制御，並列/直列変換器に供給するロードクロックと，シフトクロックの周期を拡大率に応じて変更するための外付けハードウェアが必要となります。PC-9801は，残念ながら拡大表示を行うハードウェア構成にはなっていません。

また，GDCは描画時にデータを拡大する**拡大描画機能**ももっています。拡大率は拡大表示と同様に1～16倍です。拡大描画機能を使うと，8×8ドットの文字を描画するだけで最大128×128ドットの文字まで拡大できます。しかし，拡大時にはなんの補完処理もしてくれません。単にドットが拡大されたままの，ガタガタの文字になるだけです。この拡大描画機能は，PC-9801で利用することができます。

8 自動カーサシフト

GDCは3つの表示/動作モードをもっています。

- ●文字/グラフィック混在モード
- ●文字モード
- ●グラフィックモード

このうち，文字制御時には，カーソルを表示する機能があります。カーソルはブロックカーソルやアンダーラインカーソルなどに，形状をある程度設定することができます。そして，文字を書き込むと自動的にカーソルの位置を進めてくれます。カーソルシフトの方向は左から右だけでなく，右から左，または縦方向にも行えます。

▣ 1 文字単位でのアトリビュート設定

文字制御時には文字の点滅/反転表示などが可能ですが，これらを設定する文字属性(アトリビュート)は1文字単位に設定することができます。

PC-9801の場合も1文字単位のアトリビュートとなっています。

9 256文字/行，100行/画面

GDCは1行中に最大256文字，1画面中に最大100行を表示することができるようになってます。これらの値は，自由に設定することが可能です。

10 インタレース/ノンインタレース走査

走査線の走査方式にはインタレース/ノンインタレースの2種類がありますが，GDCはこのいずれの走査方式も選択することができます。

最近のパソコンはちらつきの少ないノンインタレース走査がほとんどですが，PC-9801XL²などは，長残光性のディスプレイを使ってインタレース走査をしています。これはCRTの価格を下げるためだと思います。

■プログラマブルな同期信号設定

GDCは，水平同期信号のHFP(Horizontal Front Porch)，HS(Horizontal Sync)，C/R(Characters/Row)，垂直同期信号のVBP(Vertical Back Porch)，L/F(Lines/Frame)，VFP(Vertical Front Porch)，VS(Vertical Sync)などの幅を自由に設定できます。

2 PC-9801に固有なこと

本書ではPC-9801でのGDCの使用法を考えていきます。そのためには，GDCに関係するPC-9801のハードウェアの知識が多少必要となってきます。

1 ふたつのGDC

PC-9801はふたつのGDCを搭載しています。ひとつはテキスト用，もうひとつはグラフィック用です。テキスト用GDCはマスター動作(同期信号を自分で発生)しています。グラフィック用GDCはスレーブ動作しており，マスターGDCの発生する同期信号に同期して動作します。

2 テキストVRAM

PC-9801のテキストVRAMは，表1-1のようになっています。CPUのアドレスでいうとA0000H～A0FFFHがテキスト文字1ページで，A1000H～A1FFFH

●表1-1　テキストVRAM

GDCアドレス	CPUアドレス	HIGH b_{15} — b_8	LOW b_7 — b_0
0000	A0000	テキスト文字 1ページ	テキスト文字 1ページ
0800	A1000	テキスト文字 2ページ	テキスト文字 2ページ
1000	A2000	メモリは存在しない	属性文字 1ページ
1800	A3000		属性文字 2ページ

↑ワードアドレス　↑バイトアドレス

までがテキスト文字２ページです。BASICやMS-DOSなどは，文字を表示するときにテキスト文字１ページしか使用していません。しかし，RAMは搭載しているので，これをユーザーが自由に使用できます。また，A0000H～A0FFFHまでの領域に，CPUがデータを書き込むことで画面に文字を表示することができます。たとえば

```
1000 DEF SEG=&HA000
1010 POKE 0,ASC("A")
```

というBASICのプログラムを動かせば，画面の右上に〝A〞の文字を表示することができます。GDCは本来VRAMにデータを書き込む機能をもっていますが，PC-9801ではハードウェアの構成上，GDCでテキストVRAMに書き込むことはできません。

CPUから見たアドレスはA0000H～A1FFFHですが，GDCから見たアドレスは,それと異なります。GDCの場合，テキスト文字１ページは0000H～07FFH，テキスト文字２ページは0800H～0FFFHです。CPUはバイト単位にアドレスを数えますが，GDCはワード単位に数えていることに注意してください。GDCは16ビットで１アドレスです。

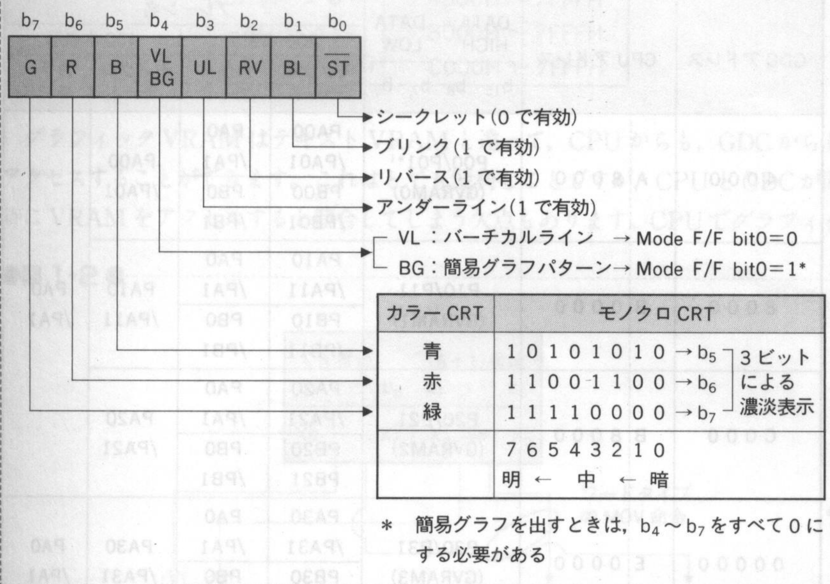
表示する文字の色/点滅/反転 などは，アトリビュートエリアにデータをセットすることで行います。アトリビュートエリアは，CPUアドレスで１ページ用がA2000H～A2FFFH，２ページ用がA3000H～A3FFFHです。なお，表1-1を見るとわかるように，メモリは下位バイトアドレスにしか存在しません。

アトリビュートエリアに書き込むデータは，図1-1のようになっています。文字色/簡易グラフ/バーチカルライン/アンダーライン/リバース/ブリンク/シークレットが指定できます。表示する文字とそのアトリビュートは１対１に対応しています。左上に〝A〞をまず表示し，それを点滅させながら赤色で表示するためには，次のようにします。

```
1000 DEF SEG=&HA000
1010 POKE 0,ASC("A")
1020 DEF SEG=&HA200
1030 POKE 0,&H43
```

もしも画面をスクロールさせる場合は，アトリビュートも同様にスクロールさせる必要があります。

●図 1-1●



アトリビュートエリアは，GDC アドレスで 1 ページ目が 1000H～17FFH，2 ページ目が 1800H～1FFFH です。

■グラフィック VRAM

PC-9801 は 128K バイトの VRAM を 2 枚もっています(表1-2)。

CPU のアドレスでいうと A8000H～BFFFFH，E0000H～E7FFFH です。これらは次の 4 つのエリアに分かれています。

A8000H～AFFFFH

B0000H～B7FFFH

B8000H～BFFFFH

E0000H～E7FFFH

本書では，これらを便宜上それぞれプレーン 0(青 VRAM)，プレーン 1 (赤 VRAM)，プレーン 2(緑 VRAM)，プレーン 3 と呼んでいます。プレーン 0 からプ

●表 1-2 プレーンの GDC アドレス

GDC アドレス	CPU アドレス	DATA HIGH (b_{15} ～ b_8) DATA LOW (b_7 ～ b_0)	200本表示 モノクロモード	200本表示 カラーモード	400本表示 モノクロモード	400本表示 カラーモード
4000	A 8000	P00/P01*1 (GVRAM0)	PA00 /PA01	PA0 /PA1	PA00 /PA01	PA0 /PA1
			PB00 /PB01	PB0 /PB1		
8000	B 0000	P10/P11 (GVRAM1)	PA10 /PA11	PA0 /PA1	PA10 /PA11	
			PB10 /PB11	PB0 /PB1		
C000	B 8000	P20/P21 (GVRAM2)	PA20 /PA21	PA0 /PA1	PA20 /PA21	
			PB20 /PB21	PB0 /PB1		
*2 オプション 00000	E 0000	P30/P31 (GVRAM3)	PA30 /PA31	PA0 /PA1	PA30 /PA31	PA0 /PA1
			PB30 /PB31	PB0 /PB1		

→ バイトのアドレス（アドレスの最下位により H，L バイトを切り分ける）
→ ワードのアドレス（H，L バイトは同時にアクセス）

＊1　上記空間において，PC-9801/U では P01，P11，P21，P31 は存在しない
＊2　8色モードのときはオプションがあってもバッファが Enable にならない
　　I/O ポート（42H）bit3＝0 のときオプションあり

レーン 2 は，初期状態でそれぞれ青/赤/緑色に対応しています。

たとえば

```
1000 DEF SEG=&HA800
1010 FOR I=0 TO &H7FFF
1020    POKE I,&HFF
1030 NEXT I
```

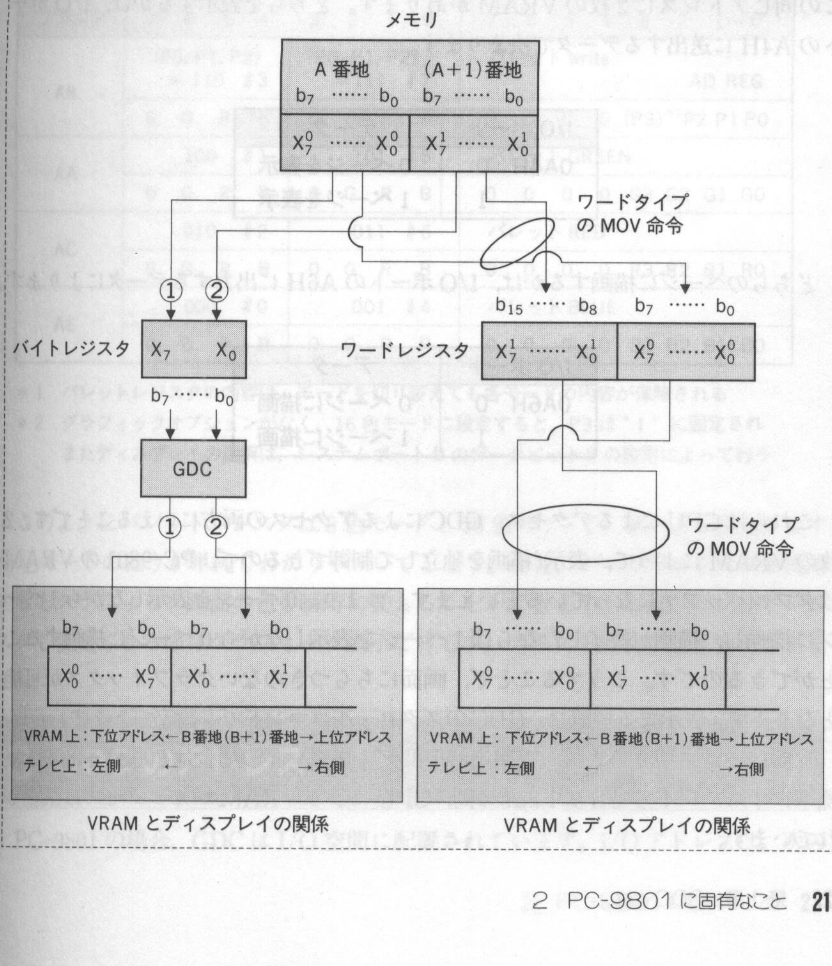
というプレーン 0 を FFH で埋めるプログラムを実行すると，画面は青色になります。

これらのプレーンの GDC アドレスは，次のとおりです。

	GDC アドレス
プレーン 0	4000H ～ 7FFFH
プレーン 1	8000H ～ 7FFFH
プレーン 2	C000H ～ 7FFFH

グラフィック VRAM はテキスト VRAM と違って，CPU からも，GDC からもアクセスすることができます。これはとても便利なことですが，CPU と GDC が同時に VRAM をアクセスすると競合してしまう欠点もあります。CPU でグラフィッ

●図 1-2●



クVRAMをアクセスする場合，GDCが描画中でないことを確認してから行う必要があります。

また，メモリをCPUからアクセスする場合とGDCからアクセスする場合とでは，ビット-ドット間の対応が逆になります(図1-2)。

CPUから直接アクセスする場合は，メモリと画面との対応がそのままです。しかし，GDCは逆になります。

■ページ切り替え

グラフィックVRAMはA8000H～BFFFFHとE0000H～E7FFFHですが，この同じアドレスに2枚のVRAMがあります。どちらを表示するかは，I/OポートのA4Hに送出するデータで決まります。

I/Oポート		データ
0A4H	0	0ページを表示
	1	1ページを表示

どちらのページに描画するかは，I/OポートのA6Hに出力するデータによります。

I/Oポート		データ
0A6H	0	0ページに描画
	1	1ページに描画

これらはCPUによるアクセス，GDCによるアクセスの両方にいえることです。2枚のVRAMによって，表示/描画を独立して制御できるので，PC-9801のVRAMはダブルバッファになっているといえます。つまり，0ページを表示しながら1ページに描画し，描画が終了したならば1ページを表示しながら0ページに描画することができるのです。こうすることで，画面にちらつきのないグラフィックスが可能となります。表示する位置は，GDCのスクロールコマンドでも設定できます。

なお，バンク切り替え後，専用高解像度ディスプレイの場合は700ns～800ns，標準ディスプレイの場合は1.2μsの間，CPUによるVRAMへのアクセスは禁止されています。

■パレット

PC-9801はパレットレジスタをもっています。パレットレジスタは一般にはルックアップテーブルといわれています。これは，VRAMに書かれたデータと実際にCRTへ表示される色の対応をさせるものです。パレットレジスタはI/OアドレスのA8H，AAH，ACH，AEHです。出力するデータと各メモリプレーンとの対応は，表1-3のとおりです。

●表 1-3 パレットレジスタ

I/O アドレス	8色モード*1								16色モード*1							
	bit 7	6	5	4	3	2	1	0	bit 7	6	5	4	3	2	1	0
A8	(P0, P1, P2) ＝110 #3				(P0, P1, P2) ＝111 #7				パレット write							AD REG
	0	G	R	B	0	G	R	B	0	0	0	0	(P3)*2	P2	P1	P0
AA	100 #1				101 #5				パレット GREEN							
	0	G	R	B	0	G	R	B	0	0	0	0	G3	G2	G1	G0
AC	010 #2				011 #6				パレット RED							
	0	G	R	B	0	G	R	B	0	0	0	0	R3	R2	R1	R0
AE	000 #0				001 #4				パレット BLUE							
	0	G	R	B	0	G	R	B	0	0	0	0	B3	B2	B1	B0

＊1　パレットレジスタの内容は，モードを切り替えても各モードの内容が保障される
＊2　グラフィックオプションがなく，16色モードに設定すると，P3は〝1〟に固定され
　　またディスプレイの選択は，システムポートBのデータビット3の設定によって行う

このようにパレットレジスタは8色モードと16色モードで，使い方が異なります。16色モードのほうが単純でわかりやすいでしょう。また，パレットレジスタを使用するとグラフィック画面の色を瞬時に変えることができます。なお，パレットレジスタで変更できるのはグラフィック表示だけです。テキストの文字色は変更できません。

3 GDCのアドレス

PC-9801の場合，GDCはI/O空間に配置されています。I/Oアドレスは，次の

とおりです。

テキストGDC	
リードステータス	60H
ライトパラメータ	60H
リードデータ	62H
ライトコマンド	62H

グラフィックGDC	
リードステータス	A0H
ライトパラメータ	A0H
リードデータ	A2H
ライトコマンド	A2H

たとえば，グラフィックGDCに対してコマンド0DHを送出するBASICプログラムは，次のようになります。

```
1000 WHILE (INP(&HA0) AND 2)<>0 : WEND
1010 OUT &HA2,&H0D
```

リスト1に，コマンドやパラメータをGDCへ送出する関数とグラフィックスの初期化関数を示します。これらの関数は，「第3章　GDCの描画例」に示した関数とリンクして使用します。

◉リスト 1

```
/*
        グラフィックス画面の初期化と
        GDCドライバ関数

        SAVE "GDCLIB.C"
*/

#include <dos.h>
#include <string.h>
#include <stddef.h>

char clear_buffer[80];

/*
        画面の表示開始
*/
void start()
{
        union REGS ireg,oreg;

        ireg.h.ah=0x42;
```

```
        ireg.h.ch=0xc0;
        int86(0x18,&ireg,&oreg);
}

/*      640*400モード
*/
void all()
{
        union REGS ireg,oreg;

        ireg.h.ah=0x40;
        int86(0x18,&ireg,&oreg);
}

/*
          画面消去
*/
void cls2()
{       int i;
        int y;

        /* ローカルバッファのクリア */
        for(i=0;i<=79;i++)
                clear_buffer[i] = 0;

        /* 各メモリプレーンのクリア */
        for(y=0;y<=399;y++){
                poke((unsigned)0xa800 , (unsigned)80*y ,
                    clear_buffer ,sizeof(clear_buffer));
                poke((unsigned)0xb000 , (unsigned)80*y ,
                    clear_buffer ,sizeof(clear_buffer));
                poke((unsigned)0xb800 , (unsigned)80*y ,
                    clear_buffer ,sizeof(clear buffer));

        }
}

/*
        GDCへのコマンド出力関数
        void send_command( cmd );
        int cmd;            送出するGDCのコマンド
*/
void send_command( cmd )
int cmd;
{
        while( (inp(0xa0) & 2) !=0 );
        outp( 0xa2 , cmd );
}
```

```
/*
        GDCへのパラメータ出力関数

        void send_parameter( pmtr );
        int pmtr;       送出するGDCのパラメータ値
*/

void send_parameter( pmtr )
int pmtr;
{
        while( (inp(0xa0) & 2) != 0 );
        outp( 0xa0 , pmtr );
}

/*
        GDCへのパラメータ(ワード)出力関数

        void send_parameter( wpmtr );
        int wpmtr;      送出するGDCのパラメータ値(ワード)
*/
void send_word_parameter( wpmtr )
int wpmtr;
{
        int p1,p2;

        p1 = wpmtr & 0xff;
        p2 = wpmtr >> 8;
        while( (inp(0xa0) & 2) != 0 );
        outp( 0xa0 , p1);
        while( (inp(0xa0) & 2) != 0 );
        outp( 0xa0 , p2);
}
/*
        VERTICAL  SYNCの起動を待つ
        関数
*/
void vsync()
{
        while((inp(0xa0) & 32) != 0);
}
```

■そのほかのI/O

テキスト表示制御関係の命令は，次のとおりです。

●表 1-4 テキスト表示制御関係の命令一覧

命令	I/Oポートアドレス	R/W	データ D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	備考
リードステータス	60	R	← GDCステータスフラグ →								
ライトパラメータ	60	W	← GDCパラメータ →								
リードデータ	62	R	← GDCデータ（ライトペン） →								
ライトコマンド	62	W	← GDCコマンド →								＊1
CRTインタラプトリセット	64	W	×	×	×	×	×	×	×	×	＊2
ライトモードレジスタ(1)	68	W	0	0	0	0	ADR2 (Mode F/F)	ADR1	ADR0	DT	
ライトボーダーカラー	6C	W	ボーダーカラー 0	G	R	B	0	0	0	0	
ライトモードレジスタ(2)	6A	W	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0	DT	16色/8色モード切り替え（下図参照）

●ライトモードレジスタ(2)

ADR 6543210	名前	DT 1	DT 0
0000000	COLOR SEL	16色モード	8色モード

ハードウェアリセット時0側（8色モード）になる

ADR＝0以外をライトすると動作は保証されない

＊1　GDC(M)は描画機能がないので，描画に関するコマンド（描画制御コマンド）は使用できない

＊2　スムーススクロール制御のためのCRTの垂直同期ごとの割り込みをリセットする。CRTVはパワーオン後1回割り込むが，リセットを出さない限り，その後は割り込まない

グラフィック制御関係の命令は，次のとおりです。

●表1-5　モードレジスタ

命　令	I/O ポートアドレス	R/W	データ D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	備　考
ライトモードレジスタ	68	W	0 0 0 0 0 0 1 0	カラーグラフックモード
ライトモードレジスタ	68	W	0 0 0 0 0 0 1 1	モノクログラフックモード
ライトモードレジスタ	68	W	0 0 0 0 1 0 0 1	専用高解像度ディスプレイの200本モード
ライトモードレジスタ	68	W	0 0 0 0 1 0 0 0	高解像度および標準ディスプレイの200本モードまたは専用高解像度ディスプレイの400本モード
ライトモードレジスタ	6A	W	0 0 0 0 0 0 0 0	8色モード
ライトモードレジスタ	6A	W	0 0 0 0 0 0 0 1	16色モード

4 I/O アクセス条件

I/Oにアクセスするとき，若干の制限があります。それは，リカバリータイムです。

連続してIN/OUTを行う場合，周辺LSIの動作が完了するのを待つ必要があります。GDCも例外ではなく，一定時間を待つ必要があります。もっともこれは，アセンブリ言語でGDCを高速に動作させる場合で，通常の使用においては，気にする必要がありません。

5 デュアルポートRAM

GDCは表示と描画の両方が行えると先に述べましたが，表示期間中に描画を行うと画面がちらつきます(フラッシュモード)。しかし，最近のPC-9801はVRAMにデュアルポートRAMを搭載し，フラッシュモードでもちらつきません。このためフラッシュモードの高速性とフラッシュレスモードの画面品質が得られるわけです。

6 GRCG と EGC

PC-9801U/UV/VF/VM には，GRCG(GRaphic CharGer)が搭載されています。PC-9801 のグラフィック VRAM は，4 プレーンに分かれていましたが，GRCG を使うと，4 画面を同時にアクセスすることができます。グラフィックの描画速度が 4 倍高速化されるわけです。

GRCG は CPU からの利用しかできませんでしたが，これを GDC からも利用できるようにしたのが EGC(Enhanced Graphic Charger)です。EGC は，PC-9801VX/RX/RA/LS などに搭載されており，GRCG との互換モードのほか，データ移動用のビット単位シフト機能/ラスターオペレーション機能ももっています。EGC の使用例は，「第 4 章　高速化」で示しています。

7 GDC のクロック

PC-9801VX/RX/RA/LS などでは，GDC の動作クロックを 2.5MHz と 5MHz に切り替えることができます。5MHz モードを使用することで，これまでの倍の描画速度が得られます。また，ディップスイッチ 2 の 8 を ON にして起動すれば，5MHz モードにすることができます。

グラフィック制御関係の命令は、次のとおりです。

4 I/Oアクセス条件

I/Oにアクセスするとき、若干の制限があります。それは、リカバリータイムです。

連続してIN/OUTを行う場合、周辺LSIの動作が完了するのを待つ必要があります。GDCも例外ではなく、一定時間を待つ必要があります。もっともこれは、アセンブリ言語でGDCを高速に動作させる場合で、通常の使用においては、気にする必要がありません。

5 デュアルポートRAM

GDCは表示と描画の両方が行えると先に述べましたが、表示期間中に描画を行うと画面がちらつきます(フラッシュモード)。しかし、最近のPC-9801はVRAMにデュアルポートRAMを搭載し、フラッシュモードでもちらつきません。このためフラッシュモードの高速性とフラッシュレスモードの画面品質が得られるわけです。

第2章

GDCのコマンド

GDCのコマンドには，次の４種類があります。

動作制御コマンド
表示制御コマンド
描画制御コマンド
VRAM制御コマンド

この章ではGDCの各コマンドを解説します。

1 動作制御コマンド

1 RESET1 コマンド

	2 進数	16 進数
	MSB　　　　　　　　LSB	
コマンド	0 0 0 0 0 0 0 0	00H

コマンド解釈に必要な時間	6 クロック
パラメータ解釈に必要な時間(1 パラメータ当たり)	2 クロック

RESET1 は，GDC を初期化するコマンドです。

GDC は RESET 端子をもっていないので，電源投入後，RESET コマンドを送出し，ソフトウェアによって初期化しなければなりません。また電源投入後，GDC の出力端子の状態は不定ですので，GDC のステータスを調べることは意味がありません。無条件に RESET コマンドを送ってください。

RESET1 コマンドが送出されると，GDC の内部では次の初期化動作が行われます。

①内部基本タイミング発生回路，同期信号発生回路，ベクタ/カウンタ回路の初期化

② FIFO アドレスのクリア，FIFO 書き込みモード設定

③描画動作，DMA 転送動作の停止

④表示の停止

⑤外部同期信号の受け付け開始(スレーブ指定時)

RESET1 は，このあとに説明する SYNC コマンドの機能ももっています。RESET1 コマンドに続いて SYNC コマンドのパラメータを送出すれば，初期化に続いて SYNC の動作を行わせることができます。電源投入直後には，RESET に続いて SYNC コマンドを送ることが多いので，このようになっているのでしょう。

●表 2-1　動作制御コマンド

コマンド	動作内容	C/P	コード							
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
RESET1	初期化動作，表示の停止，外部同期信号の受け付け開始の指示	C	0	0	0	0	0	0	0	0
RESET2	初期化動作，表示の停止，外部同期信号の受け付け停止の指示	C	0	0	0	0	0	0	0	1
RESET3*	初期化動作，表示の開始，外部同期信号の受け付け停止の指示	C	0	0	0	0	1	0	0	1
SYNC*	動作モード，同期信号波形の定義	C	0	0	0	0	1	1	1	DE
		P1	0	0	CHR	F	I	D	G	S
		P2	←			C/R				→
		P3	←	VS_L	→	←		HS		→
		P4	←			HFP		→	← VS_H	→
		P5	DS	P_H	←		HBP			→
		P6	VH	VL	←		VFP			→
		P7	←			L/F_L				→
		P8	←		VBP			→	← L/F_H	→
MASTER SLAVE	マスター動作，スレーブ動作の選択	C	0	1	1	0	1	1	1	M

＊　表示開始の指示は，動作制御コマンドおよび表示制御コマンドの初期化後実行する

RESET1 コマンド送出後，表示は停止しているので，ほかの初期化が終わったならば，START コマンドで表示を開始させてください。

2 RESET2 コマンド

	2 進数								16 進数
	MSB							LSB	
コマンド	0	0	0	0	0	0	0	1	01H

コマンド解釈に必要な時間　6 クロック
パラメータ解釈に必要な時間（1 パラメータ当たり）　2 クロック

RESET2 コマンドは RESET1 とほとんど同じコマンドですが，次の点が異なります。

　スレーブ動作時……外部同期信号を受け付けない

　マスター動作時……インタレース走査時にフィールド 1，フィールド 2 それぞれのタイミングで VSYNC を出力する

3 RESET3 コマンド

	2 進数								16 進数
	MSB							LSB	
コマンド	0	0	0	0	1	0	0	1	09H

コマンド解釈に必要な時間　6 クロック

パラメータ解釈に必要な時間（1 パラメータ当たり）　2 クロック

RESET3 コマンドも基本的に RESET1 とほとんど同じコマンドですが，次の点が異なります。

　スレーブ動作時……表示開始

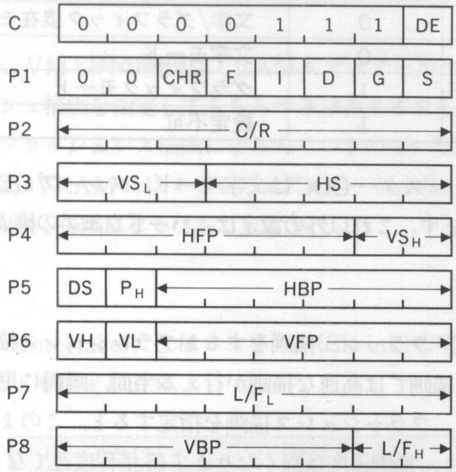
　　　　　　　　　　外部同期信号を受け付けない

　マスター動作時……インタレース走査時にフィールド 1，フィールド 2 それぞれのタイミングで VSYNC を出力

RESET2 と RESET3 コマンドの違いは，初期化後表示を開始するかどうかです。

4 SYNC コマンド

SYNC コマンドは，GDC の動作モードを決める重要なコマンドです。インタレース表示をするか，画面に横何文字表示するかなど，さまざまなモードを指定します。そのためパラメータも 8 つと多く，パラメータの意味を理解し，正しく設定するのは一苦労です。動作モードは，コマンド中の DE ビットおよびパラメータによって定義します。



コマンド解釈に必要な時間　6クロック
パラメータ解釈に必要な時間(1パラメータ当たり)　2クロック

■ DE

DE(Display Enable)は，表示を開始するか停止するかを指定します。

DE	動作
0	表示を停止
1	表示を開始

なお，表示の開始を指定する場合は，動作制御コマンド/表示制御コマンドで，あらかじめ初期化されている必要があります。さもないと，画面表示がでたらめになってしまいます。

■ CHR/G

CHRはCHaRacter，GはGraphicの意味です。GDCを文字モードで使用するかグラフィックモードで使用するか，または文字グラフィック混在モードで使用するかを指定します。

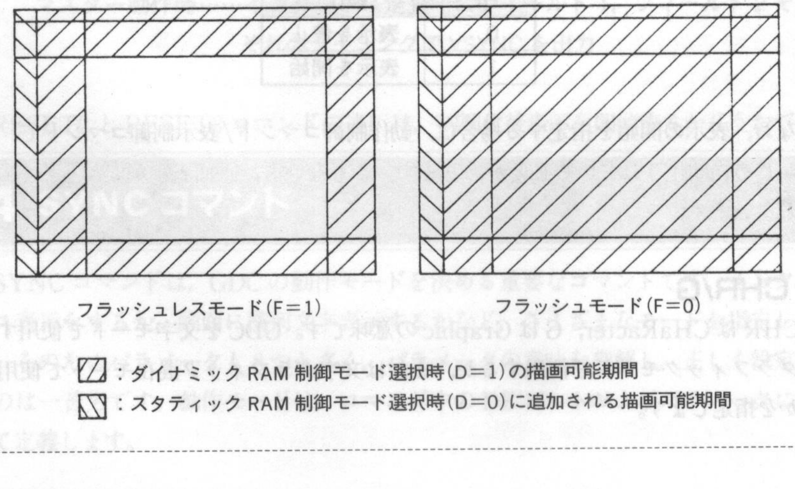
CHR	G	モード
0	0	文字/グラフィック混在モード
1	0	文字モード
0	1	グラフィックモード
1	1	設定不可

PC-9801の場合，マスターGDCは文字モード，スレーブGDCはグラフィックモードで動作しています。これ以外の設定は，ハードウェアの構成上できません。

■ F

Fは Flushの略で，フラッシュ描画をするかフラッシュレス描画をするかを指定します。フラッシュ描画では高速な描画が行える半面，画面に黒く短い横線が表示され，ちらつきます。フラッシュレス描画を指定すると，このようなちらつきは現れません。その代わり，描画速度は遅くなります(5倍程度遅くなる)。フラッシュレス描画時には，表示領域と DRAMのリフレッシュ時を除いた期間にしか，描画がされません。つまり，画面に表示している間は描画が行われないわけです。フラッシュレス描画時に描画が行われるタイミングを図2-1に示します。なお，表示をスト

●図2-1●



ップした場合は，フラッシュレス描画モードでもフラッシュ描画モードと同じ描画速度になります。

PC-9801の場合，VM以降の機種はVRAMにデュアルポートメモリを搭載しているため，フラッシュ描画を指定してもちらつきはありません。つまり，フラッシュ描画の高速さとフラッシュレス描画によるちらつきのない高品質な表示とが行えるわけです。今後は，VRAMにはすべてデュアルポートメモリが使用されるでしょうから，画面がちらつく描画はなくなるでしょう。

■ I/S

IはインタレースI(Interlace)，Sはシュリンク(Shrink)の略で，インタレース走査を行うかどうかなどを指定します。

I	S	機能
0	0	ノンインタレース
0	1	設定できない
1	0	インタレース
1	1	インタレースシュリンク

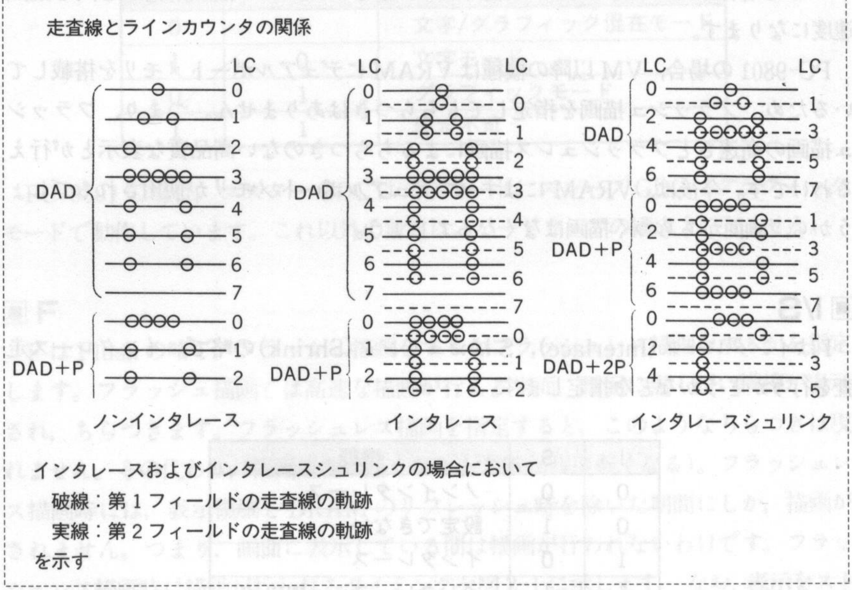
GDCでいうインタレースとは，一般にいわれているインタレースと違います。一般のインタレースは，GDCのインタレースシュリンクに相応します。

GDCでいうインタレースモードは，まず偶数本目を走査し，次に奇数本目を走査します。そのとき表示するデータは，偶数本目/奇数本目とも同じデータです。

一般でいうインタレース走査に相当するGDCのインタレースシュリンクモードは，テレビなどで使われている走査方法です。インタレースシュリンクモードには，水平周波数のあまり高くない(つまり安価な)CRTで高密度な表示ができるという利点があります。ノンインタレースに比べて，縦に倍の量の表示ができるようになっています。一方，画面が揺れているように見えるなど，目にはあまりよくありません。文字/グラフィック混在モードで画面分割を行って文字とグラフィックを混在させる場合，インタレースシュリンクモードは使用できません。

ノンインタレースモードでは，上から下まで連続的に走査します。PC-9801や最近のパーソナルコンピュータは，ほとんどノンインタレース走査です。それぞれの走査方法とデータの関係を図2-2に示します。

●図 2-2●



■ D

VRAMに使用するメモリの種類を指定します。

D	動作
0	SRAM，リフレッシュ動作なし
1	DRAM，リフレッシュ動作あり

DRAMは安価な半面，リフレッシュ動作(一定時間ごとにデータにアクセスすること)をしないとデータが変化してしまいます。一方，SRAMはリフレッシュ動作が不用で高速アクセスが可能な半面，価格が高いという欠点があります。GDCはどちらのRAMもサポートしています。DRAMの場合，このDビットを立てることによりGDCはリフレッシュ動作もしてくれるわけです。PC-9801のVRAMもDRAMにより構成されています。

■ PH

VRAMの1行が何文字で構成されるかを指定するコマンドに，PITCHがあります。PITCHで指定する文字数の第9ビット目が，このPHです。μPD7220のときはPITCHで指定できる文字数が最大256だったのですが，μPD7220Aになるときに文字数が最大512になったため，SYNCコマンドのパラメータのこんなところにPHビットが入ったようです。詳しくはPITCHコマンド(p.61)を見てください。

■ DS

DSビットは，GDC描画中にライトペンが押されたとき，描画を停止して表示アドレスを出力するか，描画を続行するかを指示します。

DS	機能
0	描画動作の一時停止なし
1	描画動作の一時停止あり

■ VH

GDCのステータスは，次のようになっています。

ビット0	DATA READY
ビット1	FIFO FULL
ビット2	FIFO EMPTY
ビット3	DRAWING
ビット4	DMA EXECUTE
ビット5	VERTICAL SYNC
ビット6	HORIZONTAL BLANK/VERTICAL BLANK
ビット7	LIGHT PEN DETECT

ステータスの第6ビットにより，水平消去信号または垂直消去信号の状態を読み出せます。このどちらを読み出すかを指定するのがVHビットです。

VH	機能
0	水平消去信号(HBLANK)
1	垂直消去信号(VBLANK)

■ VL

インタレースシュリンク走査時に，1画面当たりの走査線数を偶数にするか，奇数にするかを指定します。

VL	機能
0	奇数
1	偶数

■ C/R

画面の1行に何文字を表示するかを指定します。C/Rは，Character/Rowの略です。C/Rで指定する値と実際の値は違っているので注意してください。

C/Rで文字表示が指定できるのは2～256までの偶数です(表2-2)。C/Rが指定されるとPITCH(VRAM上での1行の文字数)もC/Rと同じ値にセットされます。そのためPITCHコマンドは，SYNCコマンドのあとに送出する必要があります。また

PITCH　≧　C/R

の関係を当然満たさなければなりません。PITCHをC/Rよりも大きく設定することにより，横方向のスクロールをさせることも可能になります。なお，SCROLLコマンドのIMビットかDAT+2ビットが立っている場合は

PITCH　≧　2＊C/R

の関係を満足させる必要があります。

●表2-2　C/R指定値

C/R	文字数
0 0 0 0 0 0 0 0	2
0 0 0 0 0 0 1 0	4
0 0 0 0 0 1 0 0	6
0 0 0 0 0 1 1 0	8
------------------------------	------
1 1 1 1 1 0 1 0	252
1 1 1 1 1 1 0 0	254
1 1 1 1 1 1 1 0	256

●表 2-3

HS					文字数
0	0	0	0	0	1
0	0	0	0	1	2
0	0	0	1	0	3
0	0	0	1	1	4
					......
1	1	1	0	1	30
1	1	1	1	0	31
1	1	1	1	1	32

▣ HS

HSはHorizontal Syncの略で，水平同期信号の幅を定義します(表2-3)。

HSには次のような制限があります。

DRAMリフレッシュ動作が必要な場合 ……………………………HSは2文字以上

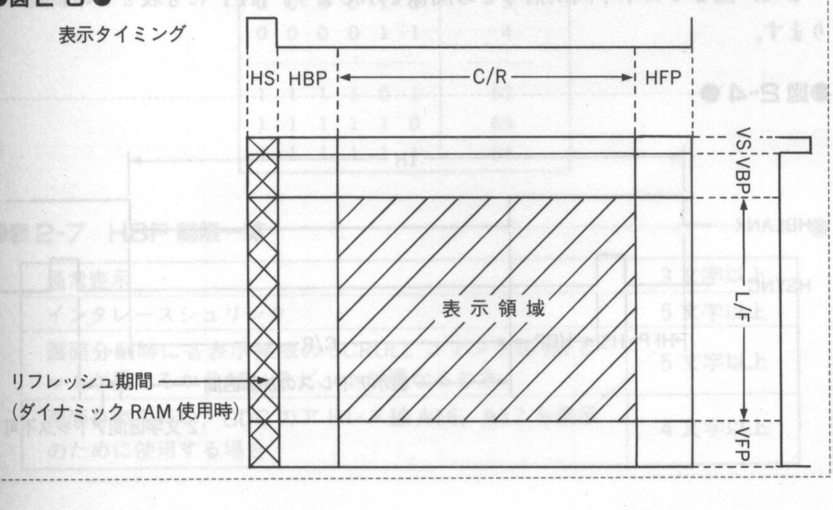
インタレースモードを指定した場合 ……………………………HSは3文字以上

インタレースシュリンクを指定した際に，GDCのアドレス線

A16，A17を表示に使う場合 …………………………………HSは5文字以上

アドレス線A16，A17を表示に使わない場合 …………………HSは4文字以上

●図 2-3●



●表 2-4

HFP						文字数
0	0	0	0	0	0	1*
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
----------						----------
1	1	1	1	0	1	62
1	1	1	1	1	0	63
1	1	1	1	1	1	64

* 設定できない。ただし，表示以外の目的に使用する場合を除く

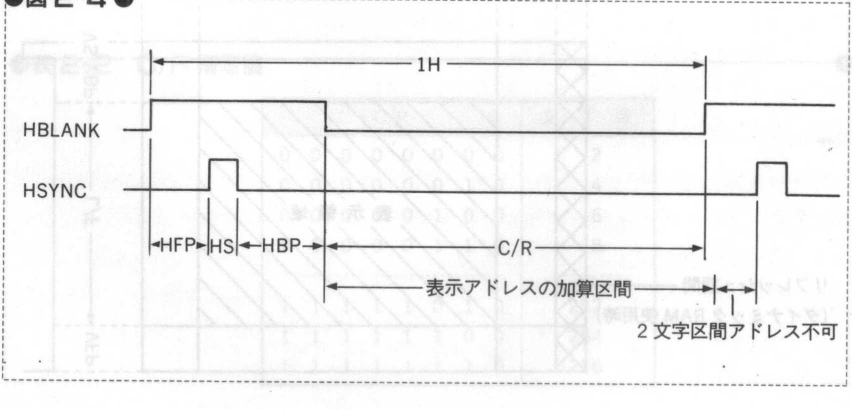
なお，DRAMのリフレッシュはHSYNC期間に行われます(図2-3)。2クロックで1アドレスがリフレッシュされるので，リフレッシュの周期を満たすようにHSの文字数を設定する必要があります。

▣ HFP

HFPは，Horizontal Front Porchの略で，画面上で右端の表示しない区間を何文字にするかを指定します(表2-4)。

また，図2-4に水平同期信号との関係を示します。HFPにも表2-5の制限があります。

●図 2-4●



●表 2-5 HFP 制限一覧

通常表示	2 文字以上
スレーブモード	CE=0 4 文字以上 CE=1 5 文字以上
ライトペン使用時	n+1 文字*
DMA 転送使用時	3 文字以上
インタレースシュリンクで拡大表示時	3 文字以上

* nはライトペン押下後，CRT に表示アドレスが表示されるまでの遅れを文字数に換算したもの

▣ HBP

HBP(Horizontal Back Porch)は，画面の左側の非表示部の文字数を設定します(表 2-6)。

HBP の制限は，表 2-7 のとおりです。

●表 2-6

HBP	文字数
0 0 0 0 0 0	1
0 0 0 0 0 1	2
0 0 0 0 1 0	3
0 0 0 0 1 1	4
------	------
1 1 1 1 0 1	62
1 1 1 1 1 0	63
1 1 1 1 1 1	64

●表 2-7 HBP 制限一覧

通常表示	3 文字以上
インタレースシュリンク	5 文字以上
画面分割時に各表示領域の SCROLL コマンドの IM ビットが異なるか DAD+2 ビットが異なるとき	5 文字以上
画面分割を行い，GDC のアドレス線 A16，A17 を表示のために使用する場合	4 文字以上

■ VS

VS(Vertical Sync)は，画面下方の非表示部の走査線数を指定します(表2-8)。

インタレースシュリンクで，1フレーム当たりの表示走査線本数を偶数に指定した場合(VL=1)，VSも偶数に指定する必要があります。

■ VBP

VBP(Vertical Back Porch)は，画面上方の非表示部の走査線数を指定します(表2-9)。

■ L/F

L/F(Lines/Field)は，1画面当たりの表示ライン数を指定します。垂直同期信号とL/Fの関係を図2-5に示します。

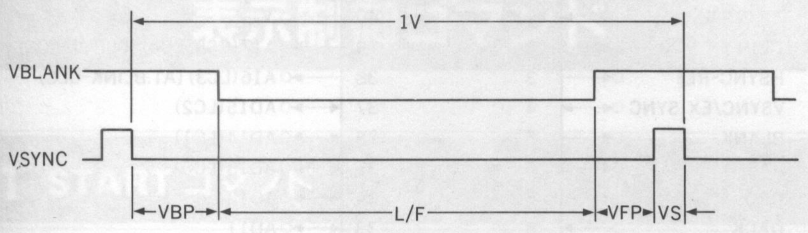
●表2-8

VS_H		VS_L			ライン数
0	0	0	0	0	設定できない
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
1	1	1	0	1	29
1	1	1	1	0	30
1	1	1	1	1	31

●表2-9

VBP						ライン数
0	0	0	0	0	0	設定できない
0	0	0	0	0	1	1
0	0	0	0	1	0	2
0	0	0	0	1	1	3
1	1	1	1	0	1	61
1	1	1	1	1	0	62
1	1	1	1	1	1	63

●図 2-5●



なお，インタレースの場合は第1フィールドの表示ライン数を設定します。インタレースシュリンクモードで1フレーム当たりの表示走査線本数を偶数に指定した場合(VL=1)，L/F も偶数に指定する必要があります。

5 MASTER/SLAVE コマンド

	2進数								16進数	
	MSB							LSB		
コマンド	0	1	1	0	1	1	1	M	6EH	スレーブ
									6FH	マスタ

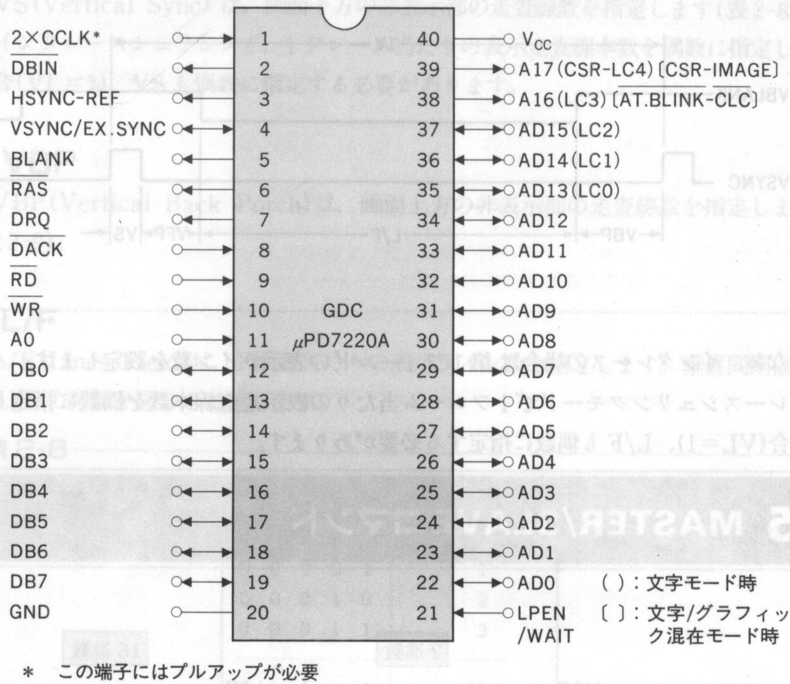
コマンド解釈に必要な時間　　12クロック

GDCをマスター動作させるか，スレーブ動作させるかを指示します。

M	動作
0	スレーブ動作
1	マスター動作

マスター動作を指示した場合，GDCのVSYNC/EX.SYNC端子はVSYNCを発生する出力端子となります(図2-6)。また，スレーブ動作時には同端子が入力端

●図2-6●



子となり，外部からの同期信号を受け付けるようになります。スレーブ動作時，EX.SYNCにテレビカメラなどの外部同期信号を入力して，外部機器と同期をとることも可能です。この機能はスーパーインポーズを行うときなどに利用します。なお，PC-9801の場合，テキストVRAMを管理するGDCがマスター動作をし，グラフィックVRAMを管理するGDCがスレーブ動作をしています。

2 表示制御コマンド

1 STARTコマンド

	MSB							LSB	
	2進数								16進数
コマンド	0	1	1	0	1	0	1	1	6BH

コマンド解釈に必要な時間　　12クロック

あるいは

	MSB							LSB	
	2進数								16進数
コマンド	0	0	0	0	1	1	0	1	0DH

コマンド解釈に必要な時間　　6クロック

STARTコマンドは，表示の開始を行います。STARTコマンドには，上記のようにふたつのフォーマットがありますが，動作は同じです。6BHのほうが12クロックと，0DHに比べて倍の実行時間がかかりますので，0DHのコマンドを使うほうがよいでしょう。6BHはμPD7220との互換性のために残されています。GDCユーザーズマニュアルには，0DHのコマンドはSTOP1コマンド(0CH)と最下位ビットしか違わないので，対にして使うと便利だと記載されていますが，ソフトウェアを作る上では，これといったメリットはないと思います。STARTコマンドを送るときは，VBLANK中でないと画面が乱れてしまいます。PC-9801のMS-DOSでは，立ち上げ時，グラフィック画面の表示を停止しています。グラフィック画面の表示を開始するには，このコマンドを送出します。

●表 2-10　表示制御コマンド

コマンド	動作内容	C/P	コード							
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
START*1*3	表示の開始の指示	C	0	1	1	0	1	0	1	1
			0	0	0	0	1	1	0	1
STOP1	表示の停止，外部同期信号の受け付け開始の指示	C	0	0	0	0	1	1	0	0
STOP2	表示の停止，外部同期信号の受け付け停止の指示	C	0	0	0	0	0	1	0	1
ZOOM	拡大表示係数，拡大描画係数の設定	C	0	1	0	0	0	1	1	0
		P1	← ZR →				← ZW →			
SCROLL	表示開始アドレス，表示領域の設定	C	0	1	1	1	← RA →			
		P1 ⋮ P16	内蔵データ RAM のフォーマットに従ってパラメータを設定							
CSRFORM	文字表示時のカーサ形状などの設定	C	0	1	0	0	1	0	1	1
		P1	CS	CE	0	← L/R →				
		P2	← BL_L →		BD	← CST →				
		P3	← CFI →					← BL_H →		
PITCH	映像メモリ水平方向ワード数の設定	C	0	1	0	0	0	1	1	1
		P1	← P_L →							
LPEN*2	ライトペンアドレスの読み出し指示	C	1	1	0	0	0	0	0	0
			← LAD_L →							
			← LAD_M →							
			×	×	×	×	×	×	← LAD_H →	

＊1　コマンドコードは6BHまたは0DHの８ビットを使用する
＊2　LPENコマンドの発行後 LAD_L，LAD_M，LAD_H，の順にCPUが読み出すことができる
＊3　表示開始の指示は，動作制御コマンドおよび表示制御コマンドの初期化後実行する

STARTコマンド，STOPコマンドはよく使うので，START＝0DH，STOP＝0CHと覚えておいても損はないでしょう。なお，STARTコマンドはGDCの動作モードをすべて設定してから実行しないと，表示がでたらめになります。

●図 2-7●

```
A:¥>SYMDEB ⏎
Microsoft (R) Symbolic Debug Utility  Version 4.00
Copyright (C) Microsoft Corp 1984, 1985.  All rights reserved.

Processor is [8086]
-FA800:0 L7FFF 33 ←------ VRAMにデータを書き込むが表示されない
-OA2 D ←----------------- START1 コマンドの送出
                          表示が開始され，画面に縦縞が現れる
-OA2 C ←----------------- STOP1 コマンドの送出。画面の縦縞が消える
-Q ←--------------------- MS-DOSへ

A:¥>
```

2 STOP1 コマンド

	2進数								16進数
	MSB							LSB	
コマンド	0	0	0	0	1	1	0	0	0CH

コマンド解釈に必要な時間　　6クロック

STOP1コマンドは表示を停止します。STOP1コマンドが送られると，GDCはBLANK信号を常にHレベルにします。また，スレーブ動作時には外部同期信号の受け付けを開始します。マスター動作でインタレース走査時には，第1フィールドを走査するときにVSYNC信号の出力を開始します。

3 STOP2 コマンド

	2進数								16進数
	MSB							LSB	
コマンド	0	0	0	0	0	1	0	1	05H

コマンド解釈に必要な時間　　6クロック

STOP2コマンドも表示を停止するコマンドです。STOP1との違いは，スレーブ動作時に外部同期信号を受け付けない点です。

4 ZOOM コマンド

	2進数								16進数
	MSB							LSB	
コマンド	0	1	0	0	0	1	1	0	46H
パラメータ	ZR				ZW				

コマンド解釈に必要な時間　　10クロック
パラメータ解釈に必要な時間(1パラメータ当たり)　　2クロック

ZOOMコマンドは，拡大表示/拡大描画時の拡大率を指定するために使います。パラメータのZRが拡大表示の拡大係数で，ZWが拡大描画の拡大係数です。ZR/ZWには，表2-11の値が設定できます。

つまり，GDCは1倍，2倍，3倍，4倍,…,16倍の拡大が行えます。拡大表示とはVRAMに書かれたデータを拡大して表示することで，VRAMの内容は変化し

●表 2-11

ZR/ZW	倍　数
0 0 0 0	1
0 0 0 1	2
0 0 1 0	3
-------------	------
1 1 0 1	14
1 1 1 0	15
1 1 1 1	16

ません。一方，拡大描画はVRAMにデータを書くときに大きくして書き込みます。なお，拡大表示は，ハードウェアも拡大表示用の構成になっていなければ行えません。PC-9801では，残念ながら拡大表示はできません。グラフィックツールなどでは拡大表示が行えると便利だと思うのですが……。拡大描画はPC-9801でも行えます。

5 SCROLLコマンド

C	0	1	1	1	←RA→

4種に画面を分割可能

コマンド解釈に必要な時間　　　10クロック

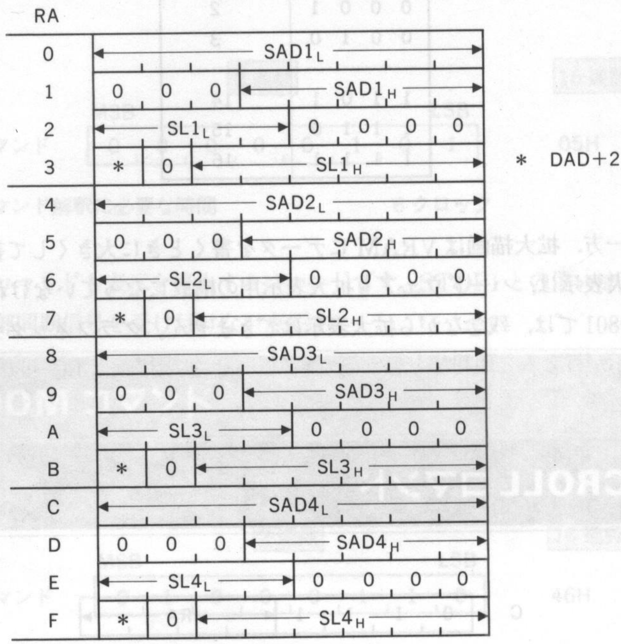
SCROLLコマンドは，VRAMのデータをスクロールさせるために使います。縦のスクロールだけでなく，横のスクロールも行うことができます。またSCROLLコマンドは，画面を分割することにも使います。分割した画面は，独立にスクロールさせることができます。このコマンドは，VRAMのどこを，画面のどこに表示するかを決定するコマンドといえるでしょう。

SCROLLコマンドのパラメータは，GDCの動作モード(文字モード，文字/グラフィック混在モード，グラフィックモード)によって違ってきます。

■文字モード

パラメータは次のようになります。

内蔵 RAM マップ



パラメータ解釈に必要な時間(1 パラメータ当たり)　　4 クロック

文字モードの場合，画面は最大 4 分割することができます。パラメータにより SAD1，SL1，SAD2，SL2，…を設定します。SAD は表示開始アドレス(GDC でのアドレス)です。SL はライン数(走査線数)です。SAD，SL と表示画面の関係は図 2-8 のようになります。SL の値とライン数の関係は，表 2-12 のようになります。

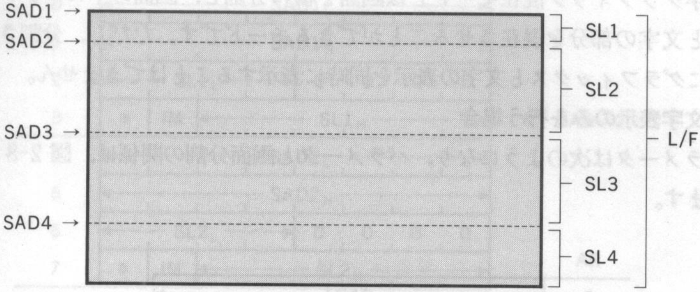
なお SL の和は，SYNC コマンドで指定した L/F(1 フレーム当たりのライン数)との間に，次のような関係が成立しなければなりません。

ノンインタレース時は

$$SL1 + SL2 + SL3 + SL4 \geqq L/F$$

です。

●図 2-8●



表示画面との対応

ただし，SAD：表示開始アドレス

SL ：画面分割表示領域の大きさを示すライン数で，SL の総和は L/F に等しいか，より大きい値にする

●表 2-12●

SL	ライン数
0 0 0 0 0 0 0 0 0 0	1024
0 0 0 0 0 0 0 0 0 1	1
0 0 0 0 0 0 0 0 1 0	2
…………	…………
1 1 1 1 1 1 1 1 1 0	1022
1 1 1 1 1 1 1 1 1 1	1023

また，文字モードでもグラフィック描画が行えます。文字モードでのグラフィック描画とは，たとえば"A"という文字で線を引くというものです(図 2-9)。

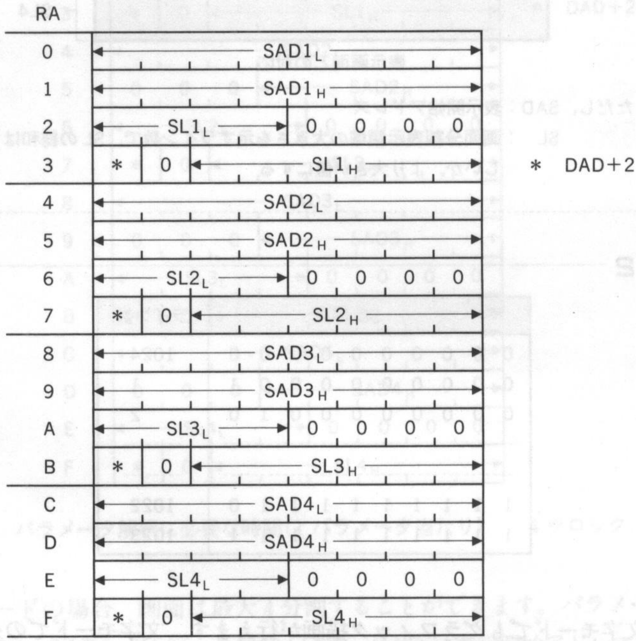
グラフィック描画を行う場合，内蔵 RAM マップ(p.52)のうち 08H ～ 0FH 番地が線種設定などに使用されるため，画面分割数はふたつまでです。なお，PC-9801 のテキスト GDC は文字モードで動作していますが，ハードウェアの構成上，グラフィック描画を行うことはできません。

■文字/グラフィック混在モード

文字グラフィック混在モードとは画面を横に分割し，画面の中にグラフィックの部分と文字の部分を混在させることができるモードです。ただし，分割された1画面中にグラフィックスと文字の表示を同時に表示することはできません。

①文字表示のみを行う場合

パラメータは次のようになり，パラメータと画面分割の関係は，図2-8のようになります。

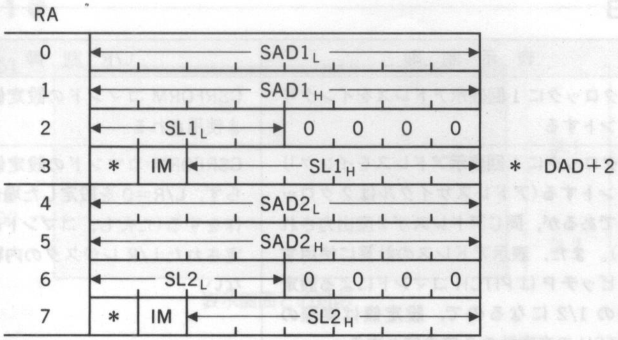


これからわかるように文字表示のみを行う場合，画面の分割数は最大4です。

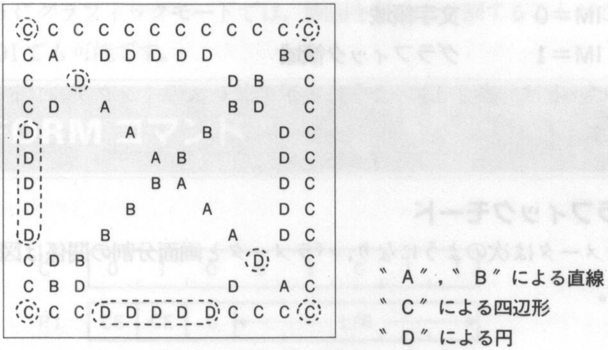
②グラフィック表示描画を行う場合

パラメータは，次のようになります。また，パラメータと画面分割の関係は図2-10のようになります。

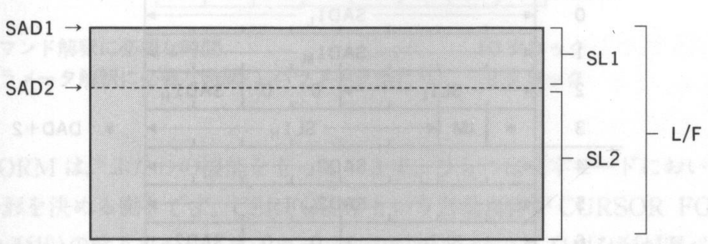
グラフィック表示描画を行う場合，画面の分割は2個までです。次のパラメータを見ると，RAMアドレスの0003Hと0007HにIMというビットがあるのが



●図 2-9●



●図 2-10●



表示画面との対応

●表 2-13

IM	表示制御	L/R 制御
0	2クロックに1回表示アドレスをインクリメントする	CSRFORM コマンドの設定値がそのまま使用される
1	4クロックに1回表示アドレスをインクリメントする(アドレスサイクルは2クロックであるが，同じアドレスが2度出力される)。また，表示アドレスの計算に使用するピッチPはPITCHコマンドによる設定値の1/2になるので，設定値は実際のPITCHの文字数の2倍の値とする	CSRFORM コマンドの設定値にかかわらず，L/R=0を設定した場合と同じ動作をする(ただし，コマンドによって設定されたL/Rレジスタの内容は変化しない)

わかります。IM(IMAGE)ビットは，分割した画面を文字領域に設定するか，グラフィック領域に設定するかを指定するのに使います。

IM=0　　文字領域

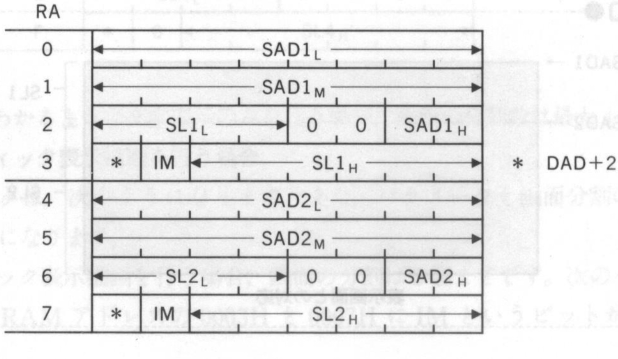
IM=1　　グラフィック領域

IMビットは，表2-13のように表示アドレスのインクリメントのタイミングを規定します。

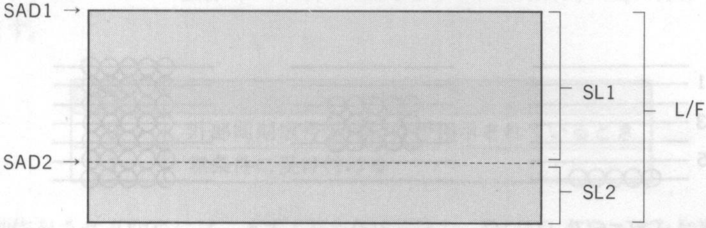
■グラフィックモード

パラメータは次のようになり，パラメータと画面分割の関係は図 2-11のようになります。

内蔵 RAM マップ



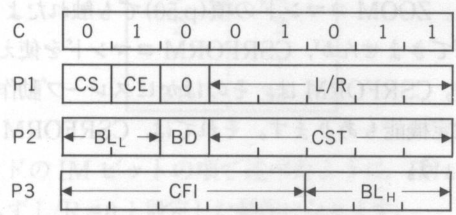
●図2-11●



表示画面との対応

グラフィックモードとは，通常のビットマップのグラフィック表示のことです。PC-9801のグラフィックGDCは，このグラフィックモードで動作しています。図2-11のようにグラフィックモードでは，画面を最大2分割することができます。これは，PC-9801でも可能です。

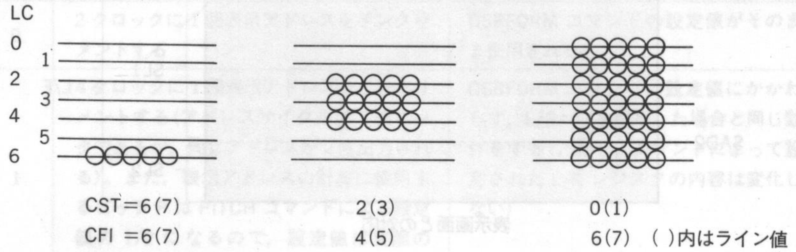
6 CSRFORMコマンド



コマンド解釈に必要な時間　10クロック
パラメータ解釈に必要な時間(1パラメータ当たり)　2クロック

CSRFORMは，ふたつの機能をもっています。ひとつは文字モードにおいてカーソルの形を決める働きです。CSRFORMという言葉自体がCURSOR FORM(カーソル形状)の略と思われます。カーソルの形状といっても自由な形が選べるわけではなく，何ライン目から何ラインをカーソルとして表示するかという指定にな

●図 2-12●



ります。したがって，表示できるカーソルはブロック状のものだけです(図 2-12)。たとえば，エディタ『MIFES』ではカーソルの形状が変更されていますが，これもおそらくCSRFORMコマンドで行っているのでしょう。

また，CSRFORMコマンドはカーソルの形のほかにカーソルを表示するかどうか，ブリンキング(点滅)させるかどうかの指定も可能になっています。

CSRFORMコマンドのもうひとつの機能とは，グラフィックモードにおいて1行を走査線何ライン分かに拡大するというものです。垂直方向の拡大表示機能と呼んでもよいでしょう。ZOOMコマンドの項(p.50)でも触れたように，PC-9801は拡大表示することができませんが，CSRFORMコマンドを使えば縦方向には拡大することができます。CSRFORMは，そのほかにスレーブ動作時の外部同期信号の受け付け条件の設定機能もあります。それでは，CSRFORMコマンドの各パラメータを見てみましょう。

■ CSビット

CSビットはカーソルを表示するかどうかを指定します。

CS	機能
0	カーソルを表示しない
1	カーソルを表示する

■ CE ビット

CE ビットは，スレーブ動作時に外部同期信号をどういう条件時に受け付けるかを指定します。

CE	機能
0	外部同期信号受け付けが指示されているとき
1	無条件に受け付ける

同期動作をさせるためには，まず CE＝0 に設定し，D1/D2 カウンタを初期化したあと CE＝1 に設定します。

■ L/R

L/R(Lines/Row)は，文字モードで 1 文字を走査線何本にするかを指定するものです。文字/グラフィック混在モードで，文字表示をする場合も設定する必要があります。グラフィックモードにおいては，通常 L/R＝0 と設定します。この L/R を意図的に変化させることにより，縦方向に最大 32 倍まで拡大表示することができます。

L/R	拡大率
0	1
1	2
2	3
⋮	⋮
31	32

SCROLL コマンドの IM ビットの項で述べたように，IM＝1 を設定すると，L/R の値にかかわらず L/R＝0 と設定した動作になります。

■ BL

BL(BLinking Rate)は，文字モードでのカーソル点滅周期を規定します。文字/グラフィック混在モードでは文字属性用タイミング信号の点滅周期となります。BL の値と点滅周期の関係は，表 2-14 のとおりです。

表 2-14 中で 1 画面とは 1V 期間を意味します。実際には BL に設定した周期の倍が周期として設定されます。カーソルが表示されている時間と消えている時間の比

●表 2-14

BL_H			BL_L		画　面
0	0	0	0	0	設定できない
0	0	0	0	1	4
0	0	0	1	0	8
0	0	0	1	1	12
0	0	1	0	0	16
0	0	1	0	1	20
0	0	1	1	0	24
------					-------
1	1	1	0	1	116
1	1	1	1	0	120
1	1	1	1	1	124

率は3：1です。たとえば，BL＝1に設定した場合，画面数は4ですから実際の周期は倍の8画面を走査するために要する時間となります。そのうち6画面を表示している間，カーソルは表示され，次の2画面を表示する間カーソルは消えています。BLは，カーソルを表示しない場合にも設定しなければなりません。

▣ BDビット

BD(Blinking Disable)は，文字モード時のカーソルの点滅を行うかどうかを規定します。

BD	動作
0	カーソルを点滅
1	カーソルを点滅しない

▣ CST

CST (Cursor StarT)は，文字モード時のカーソルを表示開始するライン位置を指定します。つまり，カーソル形状のうち上辺の位置を設定するわけです。

CSTの値とライン位置との関係は表2-15のようになります。

インタレースシュリンク時には，第1行目にカーソルを表示する場合，CST＝0としてはなりません。

●表 2-15

CST	ライン値
0 0 0 0 0	1
0 0 0 0 1	2
‥‥‥	‥‥
1 1 1 1 0	31
1 1 1 1 1	32

●表 2-16

CFI	ライン値
0 0 0 0 0	1
0 0 0 0 1	2
‥‥‥	‥‥
1 1 1 1 0	31
1 1 1 1 1	32

▣ CFI

CFI (Cursor FInish) は，文字モードでカーソルの表示を終了する位置を指定します。つまり，カーソル形状のうち下辺を何ライン目にするかの指定です。

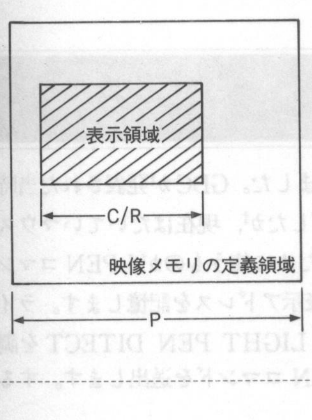
CFIの値とライン位置との関係は表2-16のようになります。

カーソルの形は，CSTとCFIを使って自由に設定できるわけです(ただし，形は長方形にしかできませんが……)。

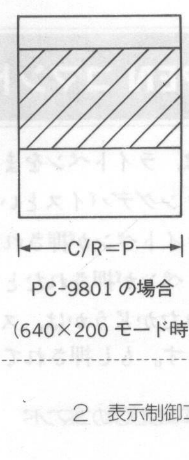
7 PITCH コマンド

PITCHコマンドは，メモリ中の横幅を定義します。これは，表示画面の横幅と同じかそれより大きく設定しなければなりません。画面の横幅より大きく設定する

●図 2-13●



●図 2-14●



ことで，横方向にもスクロールが可能になります(図 2-13)。

PC-9801 の場合，メモリ中の横幅と表示の横幅が同じに設定してあるため，あまり自由度はありません(図 2-14)。

しかし，PITCH コマンドを使って横幅を広げてやれば，横方向にもスクロールさせることができます(そのとき，縦の分解能は 200 程度に落としてやらなければ，VRAM の領域からはみ出してしまいます)。

PITCH コマンドとパラメータは，次のようになります。

C	0	1	0	0	0	1	1	1
P1	← P_L →							

P_H	P_L	文字数/行
0	0 0 0 0 0 0 0 0	0
0	0 0 0 0 0 0 0 1	1
0	0 0 0 0 0 0 1 0	2
	⋮	⋮
0	1 1 1 1 1 1 1 0	254
0	1 1 1 1 1 1 1 1	255
1	0 0 0 0 0 0 0 0	256
1	0 0 0 0 0 0 0 1	257
	⋮	⋮
1	1 1 1 1 1 1 1 0	510
1	1 1 1 1 1 1 1 1	511

コマンド解釈に必要な時間	10 クロック
パラメータ解釈に必要な時間(1 パラメータ当たり)	2 クロック

8 LPEN コマンド

最近では，ライトペンをまったく見なくなりました。GDC が発表された当時は，ポインティングデバイスといえばライトペンでしたが，現在はたいていマウスを使います。ライトペンが押された位置を検出するために使うものが LPEN コマンドです。ライトペンが押されたとき，GDC はその表示アドレスを記憶します。ライトペンが押されたかどうかは，ステータスフラグの LIGHT PEN DITECT を調べればわかります。もし押されていたならば，LPEN コマンドを送出します。すると，

GDCは表示アドレスをFIFOに転送してくれます。それをCPUで読み取ってやれば，ライトペンの押された位置に対応するVRAMアドレスを知ることができます。GDCはアドレスしか返しませんので，ドットアドレスを知る手立てはありません。

LPENコマンドと返されるライトペンアドレスは，次のようになります。

C	1	1	0	0	0	0	0	0
D1	←			LAD_L				→
D2	←			LAD_M				→
D3	×	×	×	×	×	×	LAD_H	

3 描画制御コマンド

GDCの主な機能といえるのが描画制御コマンドです。これまでの動作制御コマンド，表示制御コマンドはGDC以前のCRTコントローラにも備わっていました。CRTコントローラの機能と描画機能を合わせもっていることこそがGDCの特徴です。GDCの描画コマンドには，次のものがあります。

- VECTW コマンド
- VECTE コマンド
- TEXTW コマンド
- TEXTE コマンド
- CSRW コマンド
- CSRR コマンド
- MASK コマンド

これらのコマンドを組み合わせることにより，直線や円を描画することができるのです。

描画機能は複雑なので「第3章　GDCの描画例」で詳しく解説したいと思います。実際の使い方は，そちらを参照してください。

●表 2-17　描画制御コマンド

コマンド	動作内容	C/P	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
VECTW	描画に必要な各種パラメータの設定	C	0	1	0	0	1	1	0	0
		P1	SL	R	C	T	L	← DIR →		
		P2	← DC_L →							
		P3	×	DGD	← DC_H →					
		P4	← D_L →							
		P5	×	×	← D_H →					
		P6	← $D2_L$ →							
		P7	×	×	← $D2_H$ →					
		P8	← $D1_L$ →							
		P9	×	×	← $D1_H$ →					
		P10	← DM_L →							
		P11	×	×	← DM_H →					
VECTE	直線，四辺形，円弧，1ドット描画の実行の指示	C	0	1	1	0	1	1	0	0
TEXTW	グラフィックテキストコード設定	C	0	1	1	1	1	← RA′ →		
		P1	← TX8，PTN_L →							
		P2	← TX7，PTN_H →							
		P3	← TX6 →							
		P4	← TX5 →							
		P5	← TX4 →							
		P6	← TX3 →							
		P7	← TX2 →							
		P8	← TX1 →							
TEXTE	グラフィックテキスト描画実行指示	C	0	1	1	0	1	0	0	0
CSRW	描画アドレスの設定：文字モード	C	0	1	0	0	1	0	0	1
		P1	← EAD_L →							
		P2	0	0	0	← EAD_H →				
	文字/グラフィック混在モードで文字表示/描画	C	0	1	0	0	1	0	0	1
		P1	← EAD_L →							
		P2	← EAD_H →							
	文字/グラフィック混在モードでグラフィック表示/描画	C	0	1	0	0	1	0	0	1
		P1	← EAD_L →							
		P2	← EAD_H →							
		P3	← dAD →				WG	0	0	0
	グラフィックモード	C	0	1	0	0	1	0	0	1
		P1	← EAD_L →							
		P2	← EAD_M →							
		P3	← dAD →				WG	0	← EAD_H →	
CSRR*	描画アドレスの読み出し指示	C	1	1	1	0	0	0	0	0
			← EAD_L →							
			← EAD_M →							
			×	×	×	×	×	×	← EAD_H →	
			← dAD_L →							
			← dAD_H →							
MASK	マスクレジスタ値の設定	C	0	1	0	0	1	0	1	0
		P1	← $MASK_L$ →							
		P2	← $MASK_H$ →							

＊　CSRRコマンドの発行後 EAD_L，EAD_M，EAD_H，dAD_L，dAD_H の順にCPUが読み出すことができる

1 VECTW コマンド

VECTW コマンドは，描画機能の選択や描画に必要なレジスタの設定を行います。次に，VECTW コマンドとパラメータの形式を示します。

C	0	1	0	0	1	1	0	0
P1	SL	R	C	T	L	← DIR →		
P2	← DC_L →							
P3	×	DGD	← DC_H →					
P4	← D_L →							
P5	×	×	← D_H →					
P6	← $D2_L$ →							
P7	×	×	← $D2_H$ →					
P8	← $D1_L$ →							
P9	×	×	← $D1_H$ →					
P10	← DM_L →							
P11	×	×	← DM_H →					

■ SL・R・C・T・Lビット

これらは描画の種類を規定します。GDCで行える描画は

- ●直線描画
- ●グラフィック文字描画
- ●円，円弧
- ●四辺形

です。グラフィック文字描画とは，グラフィック画面にドットで文字を描く機能の

●表2-18

SL	R	C	T	L	
0	0	0	0	0	文字モード時の通常描画時 1ドット描画時 READ/WRITEコマンド実行時 DMA転送時
0	0	0	0	1	直線描画時
0	0	0	1	0	傾斜しないグラフィック文字描画時
0	0	1	0	0	円および弧描画時
0	1	0	0	0	四辺形描画時
1	0	0	1	0	傾斜したグラフィック文字描画時

ことです。このコマンドは，文字を傾斜させたり拡大しながら書くこともできます。

次にSL，R，C，T，Lの値と描画機能の関係を示します(表2-18)。

SL，R，C，T，Lの値と機能を比べると，SLがSLANT，RがRECTANGLE，CがCIRCLE，TがTEXT，LがLINEの略だとわかります。

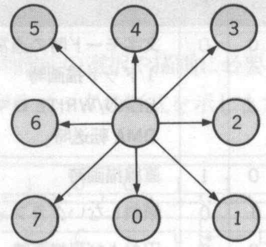
■ DIR

DIRは描画する方向を指定します。GDCは，基本的に0～45°方向の描画機能しかもちません。たとえば，直線を引く場合

0～45°
45～90°
90～135°
135～180°
180～225°
225～270°
270～315°
315～360°

の8方向に，それぞれ分けてやる必要があります。円を描画するときも同様で，中心角が45°の円弧を8回描画することで円にします。しかし，方向をプログラムで計算しなければならないことは，GDCに対して不満を感じます。GDCを改良した

●図 2-15●



AGDCでは，このような機械寄りのパラメータは，すべてAGDC内のプリプロセッサが計算してくれます。

DIRの値と方向の対応は，図2-15のようになります。

各種描画機能とDIRの関係は，表2-19のようになります。

GDCはDMAによりデータを読んだり書いたりすることができます。1次元DMAと2次元DMAができますが，1次元DMAのときと2次元DMAのときでは，転送終了時に次のようにEADの値が違ってきます(図2-16)。

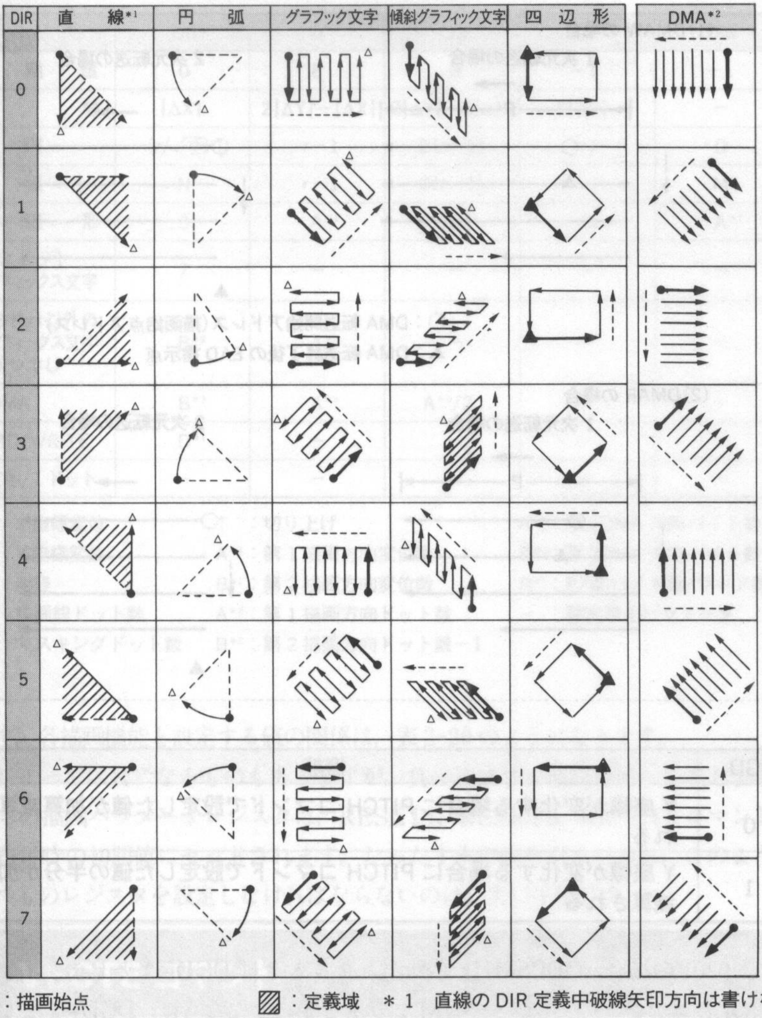
		EADの値
DMAW	1次元転送	転送開始アドレス
DMAW	2次元転送	転送終了アドレスから第1方向，第2方向に1移動した点
DMAR	1次元転送	転送開始アドレス
DMAR	2次元転送	転送終了アドレスから第1方向，第2方向に1移動した点から，さらに第1方向に1移動した点

なお，PC-9801ではGDCを使ったDMA機能は使えません。

■ DGD

DGD(Dynamic Graphic Drawing)は，描画する座標がy方向に変わる場合，それに応じて描画アドレスを更新しなければなりません。通常はPITCHコマンドで設定した値が加算/減算されますが，これを半分に変更するのがDGDビットです。

●表2-19

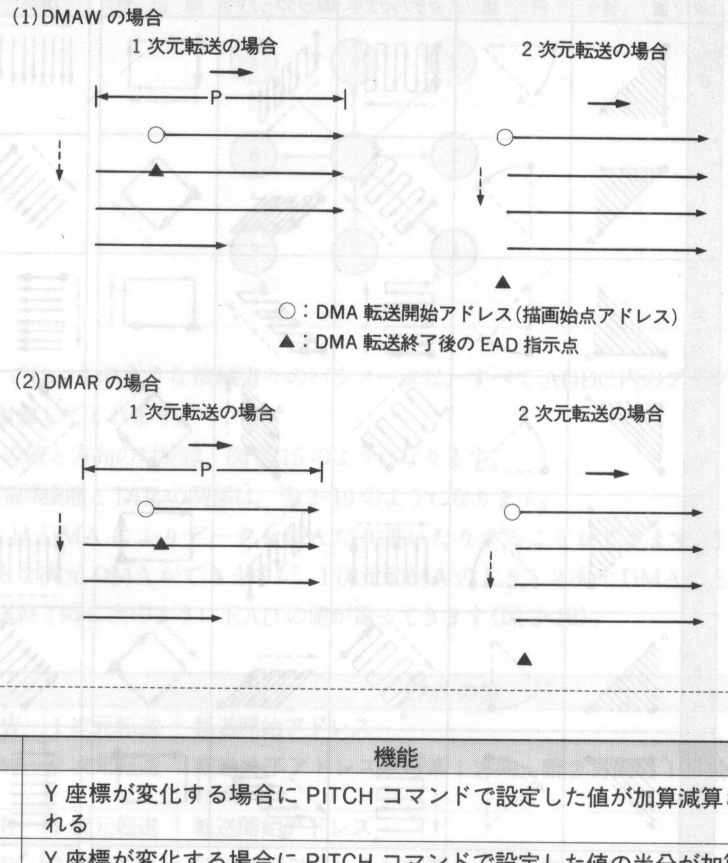


● ：描画始点
→ ：描画方向
△ ：描画終了時の EAD 指示点
➡ ：第 1 方向
- - → ：第 2 方向
▨ ：定義域
□ ：描画域

＊1　直線の DIR 定義中破線矢印方向は書けない
また，描画終了時の EAD 指示点は描画した直線の延長にくる

＊2　DMA 転送終了後の EAD 指示点は，1 次元転送/2 次元転送および DMAW/DMAR により，図 2-17 のように異なる

●図 2-16●



DGD	機能
0	Y 座標が変化する場合に PITCH コマンドで設定した値が加算減算される
1	Y 座標が変化する場合に PITCH コマンドで設定した値の半分が加算減算される

PC-9801 の場合，GDC5MHz で動作させるときは DGD=1 とします。また，SCROLL コマンドのパラメータ IM ビットを 1 にしたならば，DGD=1 としなければなりません。

■ DC・D・D2・D1・DM

これらは描画パラメータレジスタと呼ばれ，描画機能によって設定すべき値が違

●表 2-20

	DC	D	D2	D1	DM
初期値	0	8	8	−1	−1
直線	$\lvert\Delta X\rvert$	$2\lvert\Delta Y\rvert-\lvert\Delta X\rvert$	$2\lvert\Delta Y\rvert-2\lvert\Delta X\rvert$	$2\lvert\Delta Y\rvert$	−
円	$(r/\sqrt{2})\uparrow$	r−1	2(r−1)	−1	0
弧	N	r−1	2(r−1)	−1	M
四辺形	3	A*1	B*1	−1	A*1
8×8 ドット グラフィックス文字	7	−	−	−	−
8×8 ドット以外の グラフィックス文字 塗りつぶし	B*2	A*2	−	−	−
DMA	B*3	A*3	A*3/2	−	−
READ/WRITE	D*1	−	−	−	−
1文字/1ドット	−	−	−	−	−

ΔX：X 座標変位　　↑：切り上げ　　A*3：第 1DMA 方向バイト数
ΔY：Y 座標変位　　A*1：第 1 描画方向変位数　　B*3：第 2DMA 方向バイト数
r：半径　　B*1：第 2 描画方向変位数　　D*1：R/W バイト数/ワード数
N：描画総ドット数　　A*2：第 1 描画方向ドット数　　−：設定禁止パラメータ
M：マスキングドット数　　B*2：第 2 描画方向ドット数−1

います。各描画機能と設定する値の関係は，表 2-20 のようになります。

値によっては負になるものもありますが，負の値は 2 の補数を使って表します。これらの描画パラメータレジスタは，RESET1，RESET2，RESET3 コマンドや描画終了時の初期値にセットされます。たった 1 本の線を引くために，このようにいくつものレジスタを設定しなければならないのは，実に不便です。

2 VECTE コマンド

VECTE コマンドは，GDC に描画を開始させるコマンドです。VECTE コマンドを送出する前に，各種描画パラメータがセットされている必要があります。VECTE コマンドのフォーマットは，次のとおりです。

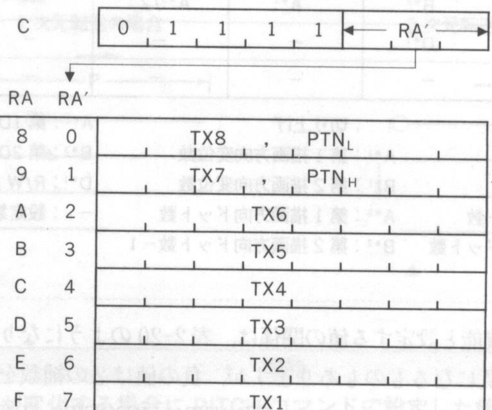
C	0	1	1	0	1	1	0	0

コマンド解釈に必要な時間　　18クロック

3 TEXTW コマンド

TEXTWコマンドは，直線や円を描画するときのラインパターン(実線，破線，一点鎖線など)を設定するときに使用します。

コマンドとパラメータは，次のとおりです。



コマンド解釈に必要な時間　　10クロック

パラメータ解釈に必要な時間(1パラメータ当たり)　　2クロック

4 TEXTE コマンド

TEXTEコマンドは，グラフィックキャラクタの描画を実行させるコマンドです。TEXTEコマンドを送出する前に，各種描画パラメータやデータRAMにフォントパターンを転送しておく必要があります。コマンドは，次のとおりです。

C	0	1	1	0	1	0	0	0

コマンド解釈に必要な時間　　16クロック

5 CSRWコマンド

CSRWコマンドは，描画実行アドレスとドットアドレスを指定します。グラフィックモードでは，このコマンドが，描画を開始する座標指定の意味になります。座標はGDCのアドレスとGDCのドットアドレス(16ドットのうち，どのドットから描画を始めるか)で指定します。X-Y座標で指定できれば，どんなに楽だったでしょう。AGDCでは，X-Y座標で指定できるようになっています。描画アドレスは一度描画を実行するとGDCの内部に保存されているので，終点から続けて描画する場合は，CSRWコマンドで指定する必要がありません。CSRWコマンドとパラメータは，次のようになります。

(1)文字モード

C	0	1	0	0	1	0	0	1
P1	← EAD_L →							
P2	0	0	0	← EAD_H →				

(2)文字/グラフィック混在モードで文字表示のみを行うとき

C	0	1	0	0	1	0	0	1
P1	← EAD_L →							
P2	← EAD_H →							

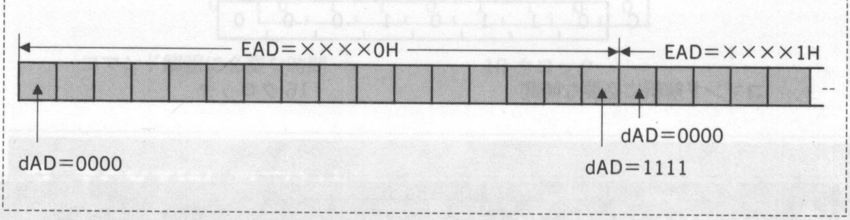
(3)文字/グラフィック混在モードでグラフィック表示/描画動作を行うとき

C	0	1	0	0	1	0	0	1
P1	← EAD_L →							
P2	← EAD_H →							
P3	← dAD →				WG	0	0	0

(4)グラフィックモード

C	0	1	0	0	1	0	0	1
P1	← EAD_L →							
P2	← EAD_M →							
P3	← dAD →				WG	0	EAD_H	

●図 2-17●



文字モード時のGDCのアドレスは，0000H-1FFFHなので13ビットで表します。また文字モードでは，ドットアドレスは意味がないので，EAD(描画アドレス)のみの指定となります。文字/グラフィック混在モードでは，アドレス可能なVRAMが0000H-0FFFFHなので，16ビットでEADを指定します。dAD(ドットアドレス)は，16ドットのうちどのドットを表すものかということなので，4ビットで表します。これは符号なしです(図2-17)。

第3パラメータ中にWGビットというのがあります。これはDGD=1のときにWRITEコマンドで描画する場合，16ビットのデータをそのままVRAMに転送するか，データの最下位ビットのみを参照するかを指定するものです。

WG	機能
0	データの最下位ビットを参照
1	データをすべて書き込む

グラフィックモードではVRAMのアドレスは，00000H-3FFFFHまで接続可能です。そのため18ビットでアドレスを指定する必要があります。

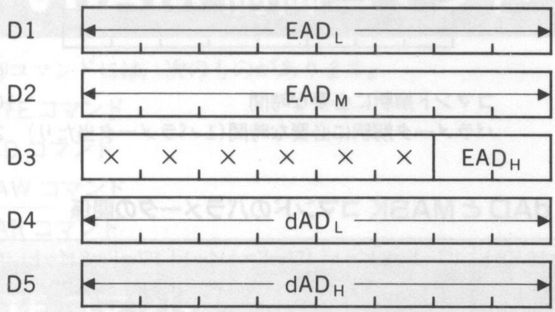
6 CSRRコマンド

CSRRコマンドは，EAD(描画アドレス)とdAD(描画ドットアドレス)を読み出すコマンドです。

C	1	1	1	0	0	0	0	0

コマンド解釈に必要な時間　　14クロック

CSRRコマンドが実行されると，GDCは次の形式でFIFOにEAD/dADを返します。



CSRRコマンドで重要な点は，GDCがFIFOにデータを返すことです。コマンド送出のタイミングとしてFIFOが空かどうかを条件に使う場合，CSRRコマンド発行後EAD/dADを読み出さないと，永久にコマンドを送出できないことになります。

また，FIFOにEAD/dADが入っている状態でコマンドを送出すれば，FIFOのデータは消えてしまいます。

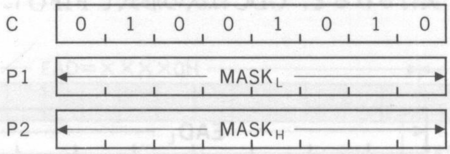
CSRRコマンドは，描画終了点を知るために使います。

7 MASKコマンド

MASKコマンドはVRAMのクリア，READコマンド時，多色同時描画時にしか使いません。VRAMのクリア時にはマスクレジスタを0FFFFHに設定します。グラフィックモードでREADコマンドを使用する場合もマスクレジスタを0FFFFHに設定します。

GDCは通常1ワードが画面の16ドットに対応しますが，1ワードを1ビット，2ビット，4ビット，8ビットにして多色表示させることもできます。その場合，設定するdADに応じてマスクレジスタの値を，表2-21のように設定する必要があります。

MASKのコマンドとパラメータは，次のとおりです。



コマンド解釈に必要な時間　10クロック
パラメータ解釈に必要な時間(1パラメータ当たり)　2クロック

●表2-21 dADとMASKコマンドのパラメータの関係

	1ワード=16ビット		1ワード=8ビット		1ワード=4ビット		1ワード=2ビット		1ワード=1ビット	
dAD	P1	P2	P1	P2	P1	P2	P1	P2	P1	P2
0	01H	00H	01H	01H	11H	11H	55H	55H	0FFH	0FFH
1	02H	00H	02H	02H	22H	22H	0AAH	0AAH		
2	04H	00H	04H	04H	44H	44H				
3	08H	00H	08H	08H	88H	88H				
4	10H	00H	10H	10H						
5	20H	00H	20H	20H						
6	40H	00H	40H	40H						
7	80H	00H	80H	80H						
8	00H	01H								
9	00H	02H								
10	00H	04H								
11	00H	08H								
12	00H	10H								
13	00H	20H								
14	00H	40H								
15	00H	80H								

MASKコマンドは特殊なコマンドなので，第3章に示した実例で使用法を確認してください。

4 VRAM制御コマンド

VRAM制御コマンドには，次のものがあります。

- WRITEコマンド
- READコマンド
- DMAWコマンド
- DMARコマンド

1 WRITEコマンド

WRITEコマンドは，2種類の機能をもちます。ひとつはGDCを通してVRAM

●表2-22　映像メモリ制御コマンド

コマンド	動作内容	C/P	コード							
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
WRITE	パラメータの映像メモリへの書き込み準備	C	0	0	1	← WLH →		0	← MOD →	
		P1 ⋮								
READ	映像メモリ/データの読み出し指示	C	1	0	1	← WLH →		0	← MOD →	
DMAW	映像メモリへのDMA転送開始の指示	C	0	0	1	← WLH →		1	← MOD →	
DMAR	映像メモリからのDMA転送開始の指示	C	1	0	1	← WLH →		1	← MOD →	

にデータを書き込む機能です。CSRW コマンドで書き込むアドレスをセットし，VECTW コマンドで SL=R=C=R=L=0 および転送バイト数を設定し，WRITE コマンドを送出します。そして，転送するデータをパラメータ送出すれば VRAM にそのデータを転送することができます。

もうひとつの機能は，ドット修正モードを変更する機能です。この場合は，単に WRITE コマンドのみを送出します。WRITE コマンドのフォーマットは，次のとおりです。

C	0	0	1	WLH	0	MOD

コマンド解釈に必要な時間		12 クロック
パラメータ解釈に必要な時間		
ワード転送時	パラメータ 1	2 クロック
	パラメータ 2	4 クロック
バイト転送時		8 クロック

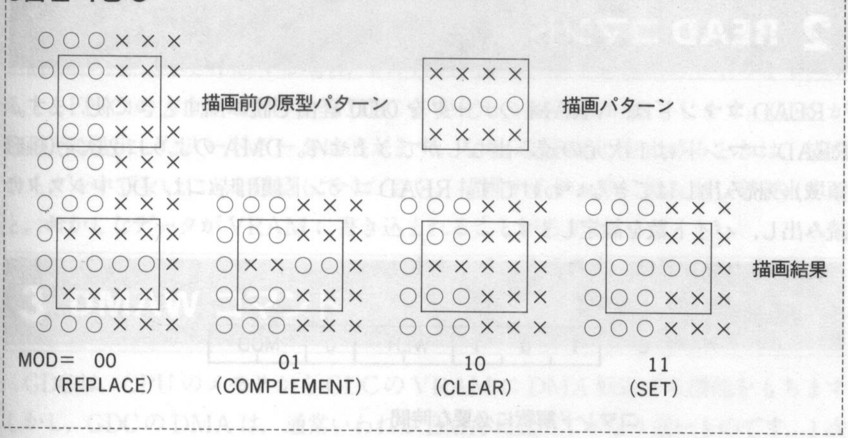
コマンドのビットパターン中に WLH と MOD があります。WLH は，転送するデータをワードにするか，上位バイトにするか，または下位バイトにするかの指定をします。

WLH	機能
00	ワード転送
01	設定不可
10	下位バイト
11	上位バイト

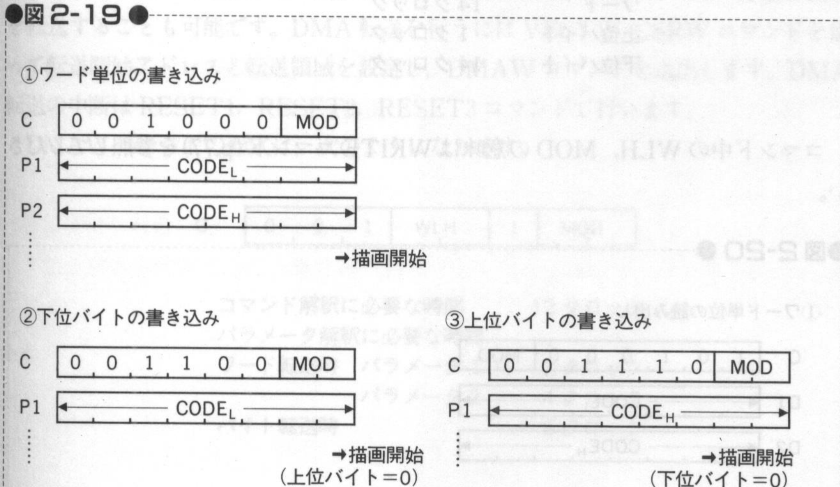
MOD はドット修正モードを指定します。ドット修正モードとは聞き慣れないことばですが，これは描画時にデータと VRAM データの AND をとるか，OR をとるかなどを指定するものです。

MOD	機能
00	REPLACE
01	COMPLEMENT
10	CLEAR
11	SET

●図2-18●



●図2-19●



機能の呼び名は描画LSIごとにまちまちですが，GDCではこのような呼び名を使っています。各ドット修正モードの具体例を図2-18に示します。描画の開始は図2-19のようになります。

2 READコマンド

READコマンドは，VRAMのデータをGDC経由で読み出すときに使います。READコマンドは1次元の読み出ししかできません。DMAのように2次元(矩形領域)の読み出しはできないわけです。READコマンド使用時には，DCレジスタに読み出し，バイト数を設定します。

READコマンドは，次のようになります。

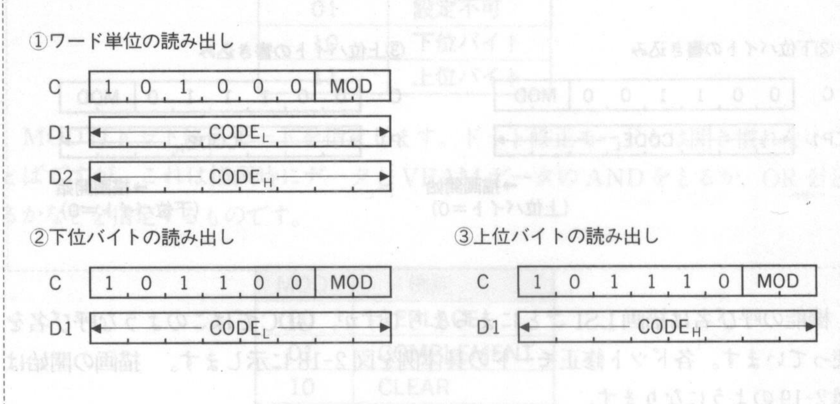
C	1	0	1	WLH	0	MOD

コマンド解釈に必要な時間

ワード	14クロック
上位バイト	1クロック
下位バイト	14クロック

コマンド中のWLH，MODの意味はWRITEコマンド(p.77)を参照してください。

●図2-20●



READコマンドを送出すると，GDCはFIFOにVRAMのデータを返してきます。これをCPUで読み出します。返されるデータの形式は，図2-20のとおりです。

読み出しを途中で中断する場合，FIFOに何かコマンドを送出してやります。すると，FIFOの方向がCPUからGDCに変化し，READが終了します。GDCがFIFOに送出したリードデータは消えます。なお，読み出しを止めるときは，描画サイクル中でないことを確認してください。描画サイクル中に読み出しを中止すると，おかしなデータがVRAMに書き込まれることがあります。

3 DMAWコマンド

GDCは，CPUのメモリからGDCのVRAMにDMA転送する機能をもちます。しかし，GDCのDMAは，通常いわれるDMAに比べてかなり遅いものです。1次元，2次元(矩形)のDMAが可能です。2次元DMA時には，45°傾いた矩形領域を転送することも可能です。DMA転送を行うにはVECTW/CSRWコマンドを使って転送開始アドレスと転送領域を設定し，DMAWコマンドを送出します。DMA転送の中断はRESET1，RESET2，RESET3コマンドで行います。

DMAWコマンドの形式は，次のようになります。

C	0	0	1	WLH	1	MOD

コマンド解釈に必要な時間		12クロック
パラメータ解釈に必要な時間		
ワード転送時	パラメータ1	2クロック
	パラメータ2	4クロック
バイト転送時		8クロック

下位バイト転送時には上位バイトが不定になります。また，上位バイト転送時には下位バイトが不定になります。これを避けるには，前出のMASKコマンドを使います。

なお，PC-9801ではGDCのDMA機能を使うことはできません。

4 DMARコマンド

DMARコマンドは，DMAによってGDCのVRAMからCPUのメインメモリに転送するときに使います。DMARコマンドを送出する前に，EADと転送領域をVECTW/CSRWコマンドで設定する必要があります。コマンドの形式は，次のとおりです。

C	1	0	1	WLH	1	MOD

コマンド解釈に必要な時間

ワード	14クロック
上位バイト	12クロック
下位バイト	14クロック

WLH，MODはWRITEコマンドと同じです。DMA動作を中断する場合はRESET1，RESET2，RESET3コマンドを使用します。

DMAリード機能もPC-9801では使えません。

GDCのコマンドは以上です。GDCのコマンドは複雑で，ほかのコマンドと合わせて使用することが多いものです。使い方は，第3章に示した実例を参照してください。

第3章

GDCの描画例

この章では，GDCを使って描画を行うプログラムを作ります。GDCはいくつものコマンドを組み合わせてはじめて描画が行えるため，コマンドの解説だけではなかなかプログラムを組めないのが実状です。この章が，GDCのプログラミングの手助けとなれば幸いです。

PC-9801はさまざまな種類が発売されています。そこで，この章ではすべてのPC-9801(XA，LTは除く)で動作するプログラムを作成します。グラフィックチャージャなどを使って高速化することもできますが，それは，第4章で紹介します。また，初期化との関係があるので，この章のプログラムはGDC2.5MHzモードで実行してください(GDCを2.5MHzにするには，ディップスイッチ2の8をOFFにして立ち上げます)。

なお，この章で紹介するプログラムは，LIFEBOAT社の『Lattice C Compiler』Ver.4.1に対応したものです。

1 直線描画

GDCの機能のうち，もっとも魅力的なものが直線描画です。従来，直線はCPUで1ドットずつ描画していました。しかし，GDCを使えばコマンドとパラメータを送出するだけで描画できます。また，800 ns/pixelという描画速度は，CPUではなかなか実現できないものです。GDCの描画は確かに高速ですが，描画を開始するまでに，計算/送出するコマンド/パラメータが結構面倒です。特にパラメータの多さには閉口させられます。この点について，AGDCでは完全に解消されています。GDCは，面倒なパラメータさえ設定すれば高速な描画という恩恵が得られますので，我慢してプログラムを書くしかありません。

直線描画を行わせるためには，次のコマンド送出が必要です。

- TEXTW コマンド
- WRITE コマンド
- CSRW コマンド
- VECTW コマンド
- VECTE コマンド

もちろん，これらの前にGDCは初期化されていることが前提となります。PC-9801の場合，GDCの初期化は，起動時にROM内ルーチンが行います。この章では，PC-9801起動時の状態で動くプログラムを考えていきます。

直前に引いた直線とラインパターンや描画モード(SET，REPLACE，COMPLEMENT，CLEARなど)が変わらない場合は

- CSRW コマンド
- VECTW コマンド
- VECTE コマンド

だけで直線を引くことができます。

それでは，これらのコマンドについて順を追って説明していきましょう。

1 TEXTW コマンド

直線の描画でTEXTWコマンドの果たす役割は，ラインパターンの設定です。ラインパターンとは実線，破線，一点鎖線などです。

コマンド	078H
パラメータ	0FFH 0FFH

上記のように，コマンド/パラメータを送出すれば，実線の直線になります。ラインパターンは，パラメータで与えるデータのビットパターンによって決まります。たとえば

15　　　　　8 7　　　　　0

●●●○○●●●○○●●●○○●

のようなラインパターンを設定したいならば，パラメータとしてE739Hを送出します。ラインパターンは，BASICのライン文と同じなので難しくはないでしょう。

ラインパターンを指定する関数textw( )は，次のようになります。

```
void textw( pattern )
int pattern;
{
        send_command( 0xaa );    /* textw ｺﾏﾝﾄﾞ */
        send_word_parameter( pattern ); /* pattern */
}
```

第2章で説明したsend_command関数とsend_word_parameter関数を使っています。ラインパターンは，GDCのパターンレジスタに保存され，新たに設定されるまでそのパターンに従って描画が行われます。TEXTWコマンドは，ラインパターンを変更したいときにだけ送出します。直線を引くたびに，毎回送出する必要はありません。不要なコマンド/パラメータの送出は時間の無駄です。GDCにコマンドを送って直線を書かせるときは，実行速度が問題になりますから，そのような無駄は極力避けるべきです。

2 WRITEコマンド

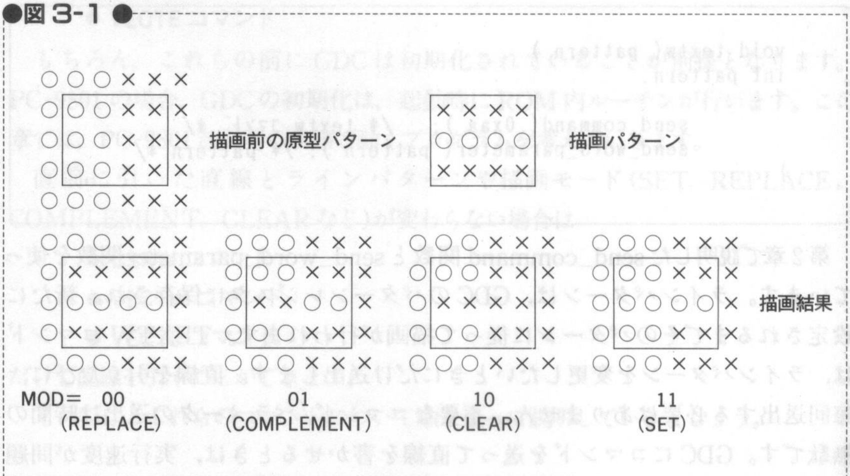
直線描画の場合，WRITEコマンドの働きは，ドット修正モードの指定にあります。ドット修正モードとは，直線を書くか直線を消すかなどの指定をするモードです。WRITEコマンドで指定できるのは

- REPLACE
- COMPLEMENT
- CLEAR
- SET

です。これは図3-1のような働きをもちます。描画はSET，消去はCLEARです。また，COMPLEMENTのようにVRAMのデータが描画されていれば消去し，消去されていれば描画するといった動作もさせられます。これはBASICのライン文にはない機能ですが，GDCに直接コマンドを送れば実行できます。ドット修正モードを設定する関数write_modeは，たとえば，次のようになります。

WRITEコマンドも，直線を描画するたびに送出する必要はありません。

●図 3-1●



```
/*
        writeコマンド送出関数
*/
#define REPLACE       0
#define COMPLEMENT    1
#define CLEAR         2
#define SET           3

void write_mode(mode)
{
        if( mode == REPLACE )           cmd=0x20;
        if( mode == COMPLEMENT )        cmd=0x21;
        if( mode == CLEAR )             cmd=0x22;
        if( mode == SET )               cmd=0x23;
        send_command( cmd );
}
```

3 CSRW コマンド

CSRW コマンドは，描画開始位置を指定するために使います。開始位置といっても X-Y 座標で指定するのではなく，VRAM のアドレスとドットアドレスで指定します。CSRW コマンドを送出する前に，X-Y 座標からアドレスを計算する必要があります。

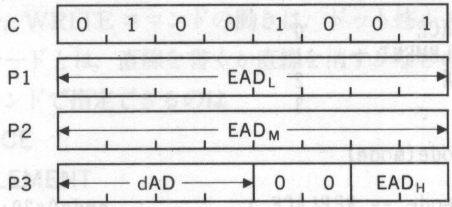
アドレスを計算するときには，画面の横幅が何文字で構成されているかを知ることが必要です。PC-9801 の初期状態では，横幅が 80 文字です。この値は，PITCH コマンドで変更できます。GDC は 1 アドレスが 16 ビットのため横 40 ワードということになります。GDC の描画開始アドレスを EAD，描画開始ドットアドレスを dAD とすれば，X-Y 座標と EAD/dAD の関係は次のとおりです。

```
ead = 0x4000 + 40 * y + x / 16
dad = x % 16
```

これは青 VRAM に対する計算式です。赤 VRAM，緑 VRAM の場合は，0x4000 のところがそれぞれ 0x8000，0xC000 に変わります。ドットアドレスの計算に注意してください。GDC では，ドットアドレスが左から右に振られているので

このような式が成り立ちます。

CSRWコマンドは，次のようにコマンド/パラメータを送出します。



PC-9801の青/赤/緑のVRAMは，GDCアドレスの4000H～FFFFHに割り当てられているので，パラメータEADHは00です。

X-Y座標を引数にとり，CSRWコマンド送出する関数csrw( )は，たとえば次のようになります。

```
void csrw( x , y )
int x,y;
{
        int ead,dad;
        int p1,p2,p3,p4

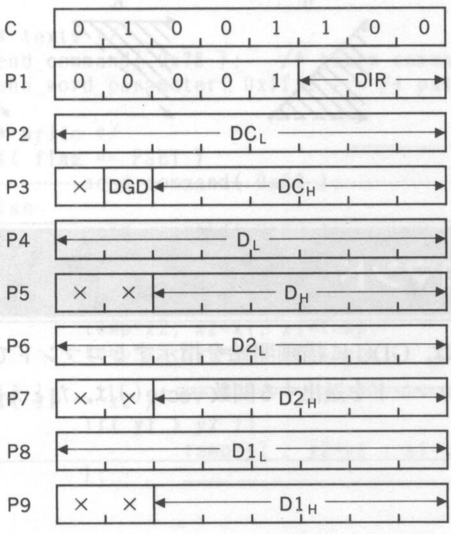
        ead     = 40 * y + x / 16;
        dad     = x % 16;

        p1      = dad<<4;
        p2      = ( ead ) & 0x00ff;
        p3      = ( ead >>  8) & 0x0ff;

        send_command( 0xaa ); /* csrw ｺﾏﾝﾄﾞ */
        send_parameter( p1 ); /* ﾊﾟﾗﾒｰﾀ 1 */
        send_parameter( p2 ); /* ﾊﾟﾗﾒｰﾀ 2 */
        send_parameter( p3 ); /* ﾊﾟﾗﾒｰﾀ 3 */
}
```

4 VECTW コマンド

VECTW コマンドは，GDC が直線描画に必要な描画パラメータを送出するのに使います。VECTW コマンドの送出は，次のとおりです。



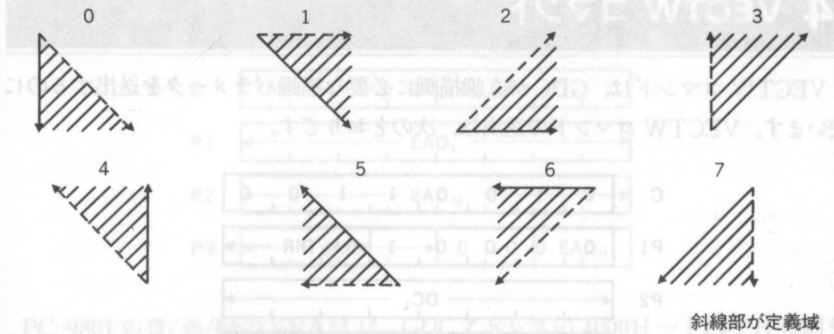
送出するパラメータの値を事前に計算する必要があります。DIR は，描画方向を表します(図 3-2)。

直線の傾きを 45° 単位で調べます。パラメータの 2 から 9 は，GDC の描画パラメータレジスタ DC，D，D2，D1 の値です。設定すべき値は，次のとおりです。

DC	D	D2	D1
$\lvert \Delta X \rvert$	$2\lvert \Delta Y \rvert - \lvert \Delta X \rvert$	$2\lvert \Delta Y \rvert - 2\lvert \Delta X \rvert$	$2\lvert \Delta Y \rvert$

VECTW コマンドは，パラメータが多く，しかも直線を描画するたびに送出しなければならないので大変です。

●図 3-2●



5 VECTE コマンド

VECTE コマンドは，GDC に描画開始を指示するコマンドです。コマンドは 6CH です。VECTE コマンドを送出する関数 vecte()は，たとえば，次のようになります。

```
void vecte()
{
        send_command( 0x6c );
}
```

以上の解説に従って作成した直線描画ルーチンを，リスト 1 に示します。これを解説していきましょう。

◉リスト 1

```
*****   ラインルーチンのリスト ******
/*
        save "cline.c"
*/
#include<dos.h>
#define PSET 0                  ─┐
#define PRESET -1               ─┘❶
```

```
void line0(x1,y1,x2,y2,plane,flag)                          ❷
unsigned x1,y1,x2,y2,plane,flag;
{
        unsigned ead,ead_offset;
        unsigned dad,dir;
        int temp;                                             ❸
        int dx,dy;
        int dc,d2,d1,d;

        /* textw */
        send_command( 0x78 );   /* textw command */          ❹
        send_word_parameter( 0xffff );  /* pattern */        ❺

        /* write */
        if( flag == PSET )
                send_command( 0x23 );                         ❻
        else
                send_command( 0x22 );

        /* init */
        if( x1 > x2 ){
                temp=x2; x2=x1; x1=temp;                      ❼
                temp=y2; y2=y1; y1=temp;
        }
        if( x1 == x2 )
                if( y1 > y2 ){                                ❽
                        temp=y2 ; y2=y1 ; y1=temp;
                };

        /* culadr */

        ead_offset=0x4000U+0x4000U*plane;
                                                              ❾
        ead = ead_offset+40*y1+x1/16;
        dad = x1 % 16;

        /* csrw */
        send_command( 0x49 );                                 ❿
        send_word_parameter( ead );                           ⓫
        send_parameter( 16*dad );                             ⓬

        /* caldir */
        dx = x2 - x1;
        dy = y2 - y1;
        if( dy > 0 ){
                if( dy > dx ){
                        dir = 0;
```

```
                                temp = dy ; dy = dx ; dx = temp;
                        }
                        else
                                dir = 1;
                }
                else{
                        dy = -dy;
                        if( dy < dx )
                                dir = 2;
                        else{
                                dir = 3;
                                temp = dy; dy = dx; dx=temp;
                        }
                }
```
⑬

```
        /* culpar */
        dc = dx;
        d = 2*dy - dx;
        d2 = 2*dy-2*dx;
        d1 = 2*dy;
```
⑭

```
        /* vectw */
        send_command( 0x4c );
        send_parameter( 8 + dir );
        send_word_parameter( dc );
        send_word_parameter( d );
        send_word_parameter( d2 );
        send_word_parameter( d1 );
```
⑮

```
        /* vecte */
        send_command( 0x6c );
}
```
⑯

```
void line(x1,y1,x2,y2,col)
unsigned x1,y1,x2,y2,col;
{
        if( (col & 1) != 0 )
                line0(x1,y1,x2,y2,0,PSET);
        else
                line0(x1,y1,x2,y2,0,PRESET);
        if( (col & 2) != 0 )
                line0(x1,y1,x2,y2,1,PSET);
        else
                line0(x1,y1,x2,y2,1,PRESET);
        if( (col & 4) != 0 )
                line0(x1,y1,x2,y2,2,PSET);
        else
                line0(x1,y1,x2,y2,2,PRESET);
}
```
⑰

```
/* lineルーチンの使用例 */
void main()
{
        int i;
        start();
        all();
        cls2();
        for(i=0;i<=639;i++)
                line(i,0,639-i,399,i % 8);
        for(i=0;i<=399;i++)
                line(639,i,0,399-i,i % 8);
}
```

解説

❶まず定数をマクロ定義します。

❷次に特定のメモリプレーンに直線を描画する関数line0を定義します。

❸使用する変数です。

❹TEXTWコマンドを送出します。TEXTWコマンドは78Hなので，send_command関数でGDCに送出します。

❺TEXTWコマンドのパラメータを送出します。この送出データで直線のパターン(実線，破線，一点鎖線など)が決まります。ここでは実線(0xffff)を送出しています。パラメータがワードなのでsend_word_parameter関数で行います。

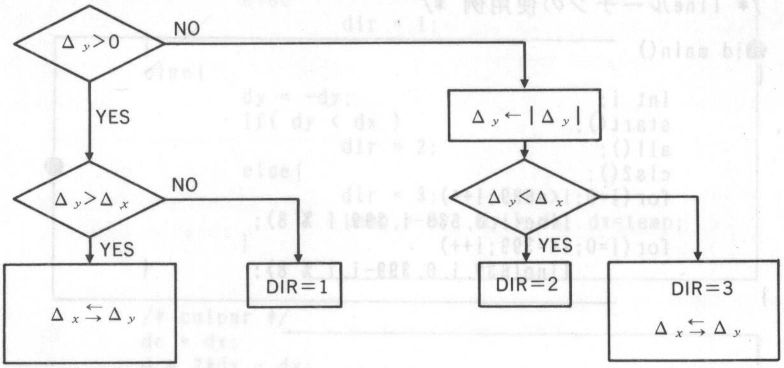
❻WRITEコマンドを送出します。直線を描画するモード(SET，REPLACE，COMPLEMENT，CLEAR)によって，WRITEコマンドを変える必要があります。BASICでいうところのPSETならば23H，PRESETならば22Hです。それに応じてコマンドを変えるプログラムを組みます。

❼CSRWコマンドです。プログラムを簡単にするために，常に$x_1<=x_2$であるような直線描画に正規化します。

❽また$x_1=x_2$の場合，$y_1>y_2$ならばY座標を入れ替えます。
本来ならば直線の描画方向は8方向のどれに該当するか計算しなければなりませんが，この前処理により場合分けを4つに減らしています。

❾次にX-Y座標から描画開始アドレス，ドットアドレス(EAD，dAD)を計算しま

●図 3-3●



す。各描画プレーンの先頭アドレスを変数ead_offsetに入れています。

⑩CSRWコマンド49Hを送出します。

⑪続いてEADをパラメータ送出します。EADはワード長のパラメータなのでsend_word_parameter関数を使います。

⑫次にdADを送出します。dADは，CSRWのパラメータの中で上4ビットなので，このようになります。

⑬次にもっともパラメータの多いコマンドVECTWです。まず描画方向を決定します。DIRの値は図3-3のフローチャートに従って決定します。

⑭次にdx，dyから描画パラメータを計算します。

⑮送出するデータが計算できたので，VECTWのコマンド/パラメータを送り出します。

⑯最後にGDCに描画開始を指示するコマンドVECTEを送出します。VECTEコマンドは6CHなので，このようになります。

⑰これで1プレーンに対し直線を描画するline0関数が定義できました。いろいろな色の直線を描くには，このline0を3回呼ぶ必要があります(EGCを使用する場合は，一度描画するだけで4画面同時に書き込むことができます。これは第4章で解説します)。

たとえば(0,0)－(639,399)を白い直線で結ぶには

```
line0(0,0,639,399,0,PSET);
line0(0,0,639,399,1,PSET);
line0(0,0,639,399,2,PSET);
```

のようにline0関数を3回呼び出します。青の場合は，プレーン0に対して描画，プレーン1，プレーン2に対して消去を指定する必要があります。すなわち

```
line0(0,0,639,399,0,PSET);
line0(0,0,639,399,1,PRESET);
line0(0,0,639,399,2,PRESET);
```

これを行う一般的なline関数が，このような表記になります。

⑱指定した色コードを変数colに代入し，colのビットパターンをテストして，PSETとするかPRESETとするかを決めています。このline関数を使ったデモプログラムは，画面全体に直線を描画するものです。start()，all()，cls2()は第2章で紹介しましたが，グラフィック画面の表示を開始したり640×400モードにしたりする画面消去を行う関数です。

普通の直線描画にはこのline関数を使います。

line()関数の中で内部的に使用していたline0関数は，使い方によってはおもしろい働きをしてくれます。line0()関数は，あるVRAM1プレーンに対してのみ描画する関数でした。しかし，この関数を使うと，line()関数ではできないCOMPLEMENTモードでの描画が可能です。COMPLEMENTモードでの描画は，BASICにはありません。COMPLEMENTモードは，VRAMのデータを反転させながら描画するものです。

line0()関数のCOMPLEMENTモードを使ったデモプログラムを次に示します。自分自身が描画したデータを反転させながら描画するため，複雑な模様が表示されます。そして描画し終わると，画面は完全に元の状態に戻る不思議なプログラムです。描画しかしていないのに，最後には画面がクリアされてしまいます。このようなプログラムは，GDCを直接制御しているからこそ行えるものです。

◉リスト2

```
/*
    COMPLEMENTの描画の例
    SAVE "COMPDEMO.C"
*/
#include <dos.h>
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2

void line0(x1,y1,x2,y2,plane,flag)
unsigned x1,y1,x2,y2,plane,flag;
{
    unsigned ead,ead_offset;
    unsigned dad,dir;
    int temp;
    int dx,dy;
    int dc,d2,d1,d;

    /* textw */
    send_command( 0x78 );           /* textw command */
    send_word_parameter( 0xffff );        /* pattern */

    /* write */
    if( flag == PSET )
        send_command( 0x23 );
    if( flag == PRESET )
        send_command( 0x22 );
    if( flag == COMPLEMENT )
        send_command( 0x21 );
    /* init */
    if( x1 > x2 ){
        temp=x2; x2=x1; x1=temp;
        temp=y2; y2=y1; y1=temp;
    }
    if( x1 == x2 )
        if( y1 > y2 ){
                temp=y2 ; y2=y1 ; y1=temp;
        };

    /* culadr */

    ead_offset=0x4000U+0x4000U*plane;

    ead = ead_offset+40*y1+x1/16;
    dad = x1 % 16;
```

```
    /* csrw */
    send_command( 0x49 );
    send_word_parameter( ead );
    send_parameter( 16*dad );

    /* caldir */
    dx = x2 - x1;
    dy = y2 - y1;
    if( dy > 0 ){
        if( dy > dx ){
            dir = 0;
            temp = dy ; dy = dx ; dx = temp;
        }
        else
            dir = 1;
    }
    else{
        dy = -dy;
        if( dy < dx )
            dir = 2;
        else{
            dir = 3;
            temp = dy; dy = dx; dx=temp;
        }
    }

    /* culpar */
    dc = dx;
    d = 2*dy - dx;
    d2 = 2*dy-2*dx;
    d1 = 2*dy;

    /* vectw */
    send_command( 0x4c );
    send_parameter( 8 + dir );
    send_word_parameter( dc );
    send_word_parameter( d );
    send_word_parameter( d2 );
    send_word_parameter( d1 );

    /* vecte */
    send_command( 0x6c );
}

void pattern( vx1, vy1, vx2, vy2 )
int vx1, vy1, vx2, vy2;
{
    int x1, y1, x2, y2;
    int sx1, sy1, sx2, sy2;
```

```
    x1=0;y1=0;x2=0;y2=0;
    for(;;){
        x1 = ((x1+vx1) & 0x1ff);
        y1 = ((y1+vy1) & 0x1ff);
        x2 = ((x2+vx2) & 0x1ff);
        y2 = ((y2+vy2) & 0x1ff);

        sx1=abs( x1 - 256);
        sy1=abs( y1 - 256);
        sx2=abs( x2 - 256);
        sy2=abs( y2 - 256);

        line0(sx1+192,sy1+72,sx2+192,sy2+72,0,COMPLEMENT);
        line0((256-sx1)+192,sy1+72,(256-sx2)+192,sy2+72,0,COMPLEMENT);
        line0(sx1+192,(256-sy1)+72,sx2+192,(256-sy2)+72,0,COMPLEMENT);
        line0((256-sx1)+192,(256-sy1)+72,(256-sx2)+192,(256-sy2)+72,0,COMPLEMENT);

        if( (x1==0) && (y1==0) && (x2==0) && (y2==0) ) return;
    }
}

void main()
{
    start();
    all();
    cls2();

    pattern( 0,1,1,0 );
    pattern( 0,1,2,3 );
    pattern( 0,1,3,2 );
    pattern( 0,2,1,3 );
    pattern( 1,2,2,3 );
    pattern( 2,1,4,3 );
}
```

2 円描画

GDCでは，円を描画することができますが，真円しか描画できません。画面の縦横比が1対1でない場合は，楕円になってしまいます。PC-9801は640×400モードのとき，画面の縦横比は1対1ですが，640×200モードのときは2対1になっています。つまり，この円描画は640×400のときにしか意味がないわけです。GDCは，縦と横との比率を考えながら描画をしてくれるわけではないのです。またGDCは一度に中心角が45°の円弧しか描けないので，円を描くためには8回描画をしなければなりません。プログラムを組む側からいえば少々面倒です。

また，描画領域の範囲指定はありません。画面からはみ出すかどうかを，GDCは何も考えてくれないのです。描画領域からはみ出す部分をカットする処理を**クリッピング**といいますが，最近の描画LSIは，クリッピングエリアを指定でき，便利になっています。しかし，GDCにおいてのクリッピングは，プログラマが処理すべき問題です。クリッピング処理をソフトウェアで行うと描画前処理が増え，GDCがいかに高速に描画してくれてもプログラム全体での速度は落ちてしまいます。また，円のクリッピングはなかなか手間のかかるものです。

この節では真円を描くことのみ考えます。円は画面中に完全に入っているものとし，クリッピングは考えていませんが，それでも結構複雑なプログラムになります。なかなか描画速度が速いものになっていますので，我慢してプログラミングしてください。

円描画を行わせるためには次のコマンドの送出が必要です。

- TEXTW コマンド
- WRITE コマンド
- CSRW コマンド
- VECTW コマンド
- VECTE コマンド

コマンドの種類としては，直線描画とまったく同じです。ただし，パラメータの意味が違っているものもあります。

1 TEXTW コマンド

直線描画と同じように，円をどのようなラインパターンで描画するかを指定します。このコマンドを使用すると，円を実線や破線，一点鎖線で描くことができるわけです。しかし実用上，実線と破線ぐらいしか使用しないことになるでしょう。というのは，円を8回に分けて描画するからです。きれいなラインパターンを指定しても，接続部で不連続になってしまいます。実線の場合，送出するコマンドとパラメータは，次のとおりです。

コマンド	0AAH
パラメータ	0FFH
	0FFH

2 WRITE コマンド

WRITEコマンドも直線描画の場合と同じように，円描画時のドット修正モードを指定します。SET，CLEAR(BASICのPSET，PRESETに相当する)COMPLEMENT，REPLACEが使用できます。直線の項で触れたVRAMのデータを反転させながら描画する方法は，円の場合でも使用できます。ただしこの場合，8分の1円の接続部で重複して描画すると，自分自身で描画したVRAMデータをも反転させてしまいます(図3-4)。

この場合は，SETとCLEARモードを主に使うことになるでしょう。

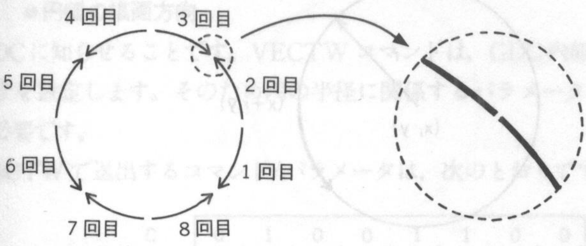
WRITEで送出すべきコマンドは，次のとおりです。

20H	REPLACEモード時
21H	COMPLEMENTモード時
22H	CLEARモード時
23H	SETモード時

WRITEコマンドに付随するパラメータはありません。

TEXTW，WRITEコマンドのふたつは，円を描画するルーチンの先頭で一度だ

●図 3-4●



円はこのように８回の描画により完成する

COMPLEMENT モードで，たとえば２回目と３回目の描画時に同じ点を重複して描画すると，その点だけ二度反転されてしまう。結局８個穴のあいた円ができあがる

けしか送出する必要はありません。無駄な送出は，プログラム実行速度を低下させます。

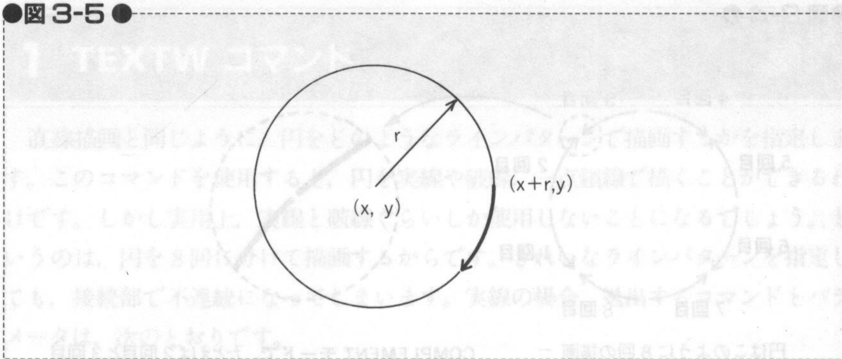
3 CSRW コマンド

CSRW コマンドは，円弧の描画開始位置を指定します。GDC は８分の１円しか一度に描画できませんが，このコマンドは，その８分の１円の描画を開始する点のアドレスを GDC に知らせるわけです。円を描画する関数は

```
void circle(x,y,r,color);
int x;          /* X座標 */
int y;          /* Y座標 */
int r;          /* 半径 */
int color;      /* 色 */
```

と定義するのが一般的でしょう。円の位置の指定には中心座標を使います。ところが，CSRW コマンドが必要としているのは，円弧の描画開始アドレスです。たとえば，図 3-5 のような円弧を描く場合，描画開始座標は

●図 3-5 ●



(x+r, y)

となります。この点のGDCから見たアドレスとドットアドレスEAD/dADは

```
ead = 0x4000 + 40 * y + ( x + r ) / 16;
dad = ( x + r ) % 16;
```

です(青VRAMに描画する場合)。CSRWコマンドの送出時には，このような中心座標から描画開始アドレス(ドットアドレス)を計算する描画前処理が必要となります。

CSRWで送出するコマンド/パラメータは，次のとおりです。

	2進数								16進数
コマンド	0	1	0	0	1	0	0	1	49H
パラメータ1	← EAD_L →								
パラメータ2	← EAD_M →								
パラメータ3	← dAD →				WG	0	EAD_H		この節では WG=0, EAD_H=00

CSRWコマンドは都合8回送出するわけですが，CSRWコマンドのパラメータ計算は，後で述べるような工夫をすれば4回で済ますことができます。

4 VECTWコマンド

円描画においてVECTWコマンドの果たす役割は

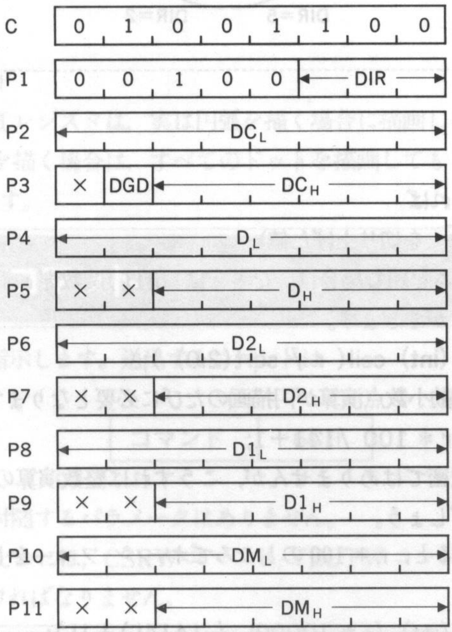
●描画の種類が円であること

●円の半径

●円弧の描画方向

をGDCに知らせることです。VECTWコマンドは，GDC内部の描画パラメータレジスタを設定します。そのため円の半径に関係するパラメータを計算する描画前処理が必要です。

VECTWで送出するコマンド/パラメータは，次のとおりです。

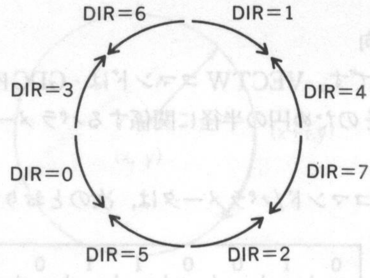


パラメータについては，次に示す値を設定します。

■ DIR

円弧を描画する方向を与えます。描画方向とDIRに与える値は，図3-6に示したとおりです。

●図 3-6●



■ DCレジスタ

円の半径をｒとすれば

DC＝(r/$\sqrt{2}$ を切り上げた値)

です。Cでプログラムを組む場合は，たとえばsqrt( )関数とceil( )関数を使うと，次のように書くことができます。

```
dc_reg = (int) ceil( r / sqrt(2.0) );
```

しかし，これでは浮動小数点演算が円描画のたびに必要となります。そこで

```
dc_reg = r * 100 /141+1;
```

とします。あまり厳密ではありませんが，こうすれば整数演算の範囲で済みます。実用上は問題ないでしょう。

半径ｒが大きくなると，ｒ＊100のところでオーバーフローをしてしまいます。それを避けたければ

```
dc_reg = (int) (r * 10000L / 1414L)+1L);
```

とロングを使って計算すればよいでしょう。このあたりは，プログラムが使用される状況に応じて変更してください。実行速度と汎用性とは，裏腹なもののようです。

■ Dレジスタ

D = r - 1

を設定します。

■ D2 レジスタ

D2 = 2＊(r − 1)

を設定します。

■ D1 レジスタ

D1 = −1

を設定します。

■ DM レジスタ

DM = 0

を設定します。DM レジスタは，実は円弧を描く場合に描画しないドット数を指定するものです。円を描く場合は，すべてのドットを描画してもらわなければ困るので，DM=0 とします。

5 VECTE コマンド

円描画の開始を指示します。送出するコマンドは

コマンド	6CH

です。VECTE に付随するパラメータはありません。

円をひとつ描くためには，CSRW コマンド/VECTW コマンド/VECTE コマンドを 8 回送出しなければなりません。

6 8 分の 1 円を描く

まず 8 分の 1 円を描く方法を考えます。図 3-5 のような円弧を描いてみましょう。プログラムはリスト 3 のようになります。

●リスト３

```
/*
  8分の1円の描画
*/
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2

void circle0( x_c,y_c,radius,plane,flag )
int x_c;                    /* x-coordinate */
int y_c;                    /* y-coordinate */
int radius;                 /* 半径 */
int plane;        /* 描画，消去するVRAMのプレーン番号 */
int flag;         /* 描画のモードを指定するフラグ */
{
    unsigned ead;
    unsigned ead_offset;
    unsigned dad;
    int xcrd;
    int ycrd;
    int dir;
    int dc_reg;         /* DC  レジスタ */
    int d_reg;          /* D   レジスタ */
    int d2_reg;         /* D2  レジスタ */
    int d1_reg;         /* D1  レジスタ */
    int dm_reg;         /* DM  レジスタ */

    /* textw */
    send_command( 0x78 );  /* textw command */
    send_word_parameter( 0xffff );  /* pattern */

    /* write */
    if( flag == PSET )
            send_command( 0x23 );
    if( flag == PRESET )
            send_command( 0x22 );
    if( flag == COMPLEMENT )
            send_command( 0x21 );

    /* vectw data */
    dc_reg = radius*100 /141+1;
    d_reg = radius-1;
    d2_reg = 2*d_reg;
    d1_reg = -1;
    dm_reg = 0;
```

（右欄外の括弧: ❶ #define行、❷ 引数宣言、❸ ead〜dad、❹ xcrd・ycrd、❺ dir〜dm_reg、❻ textw、❼ write、❽ vectw data）

```
    /* dir 7 & 4 */
        xcrd = x_c + radius;
        ycrd = y_c;                                    ❾

        /* csrw */
        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*ycrd+xcrd / 16;            ❿
        dad = xcrd % 16;

        send_command( 0x49 );
        send_word_parameter( ead );                    ⓫
        send_parameter( dad*16 );

        /* vectw */
        dir=7;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );                 ⓬
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );                          ⓭
}
/*
 circle0()関数のチェックを行なう
*/
void main()
{                                                      ⓮
        start();
        all();
        cls2();
        circle0( 320 , 200 , 100 , 0 , PSET);
}
```

解説

❶定数をマクロ定義します。

❷8分の1円を，VRAMの1プレーンに描画する関数circle0( )を定義します。

❸仮引数には中心座標(x_c，y_c)，半径radius，描画するVRAMプレーン番号plane，描画するドット修正モードflagを指定します。以下は変数宣言です。GDCから見た描画開始アドレスをead，描画ドットアドレスをdad，描画を開始するオフセットアドレスを，ead_offsetにそれぞれ入れることにします。変数ead_offsetには描画するVRAMの番号によって4000H，8000H，C000Hを代入します。

❹描画を開始する座標をxcrd，ycrdという変数に代入します。これらは中心座標と半径から計算されます。

❺GDCの描画パラメータレジスタの値を計算し，保持する変数です。

❻まず円のラインパターンを定義するために，TEXTWコマンドを送出します。TEXTWコマンドは78Hです。描画パターンは実線にします。そのためパラメータは0FFFFHです。0FFFFHはワード長であるためsend_word_parameter( )関数を使用します。

❼次に，WRITEコマンドを送出してドット修正モードを指定します。WRITEコマンドは，ドット修正モードによって20H，21H，22H，23Hとなります。仮引数flagの値によって，このように選択し送出します。

❽VECTWコマンド用のデータ，すなわち描画パラメータレジスタの値を計算します。DCレジスタの値の計算に注意してください。なお，この計算をさらに速く行うとすればシフトを使う方法もあります。

```
dc_reg = radius * 100/141+1;
```

は

```
dc_reg = radius * 128/181+1;
```

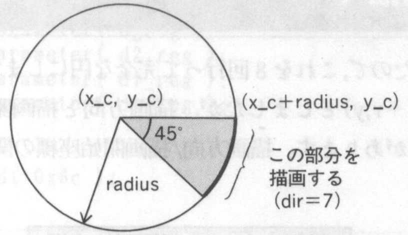
とそれほど違いはありません。だから

```
dc_reg = ( radius<<6 ) / 181+1;
```

のように〝* 128″を〝<<6″とシフトに変更すれば，多少速いコードとなるでしょう。こういったことはアセンブリ言語で記述するときにかなり有効です。

また，D2レジスタは2 *(r−1)を設定しますが，r−1の値はDレジスタに設定するので，すでに計算済みです。そこで，D2には2 * d_regを代入しています。

●図 3-7●



❾描画開始座標を計算します。いま描こうとしているのは，図3-7のような８分の１円なので描画開始座標は(x_c＋radius，y_c)となります。これを変数xcrd, ycrdに代入しています。

❿いま計算した描画開始座標から描画開始アドレス，描画開始ドットアドレスを計算します。

⓫これらをCSRWのコマンド/パラメータとして送出します。eadはワード長のパラメータなので，send_word_parameter( )関数を使います。CSRWコマンドは49Hです。

⓬次にVECTWコマンドを送出します。描画すべき円弧の方向は7なので，dir＝7となります。VECTWコマンドは4CHです。

⓭DC，D，D2，D1，DMはワード長なので，send_word_parameter( )関数を使っています。

最後に，VECTEコマンドを送出して，GDCに描画を開始させます。VECTEコマンドは6CHです。

⓮定義したcircle0( )関数のチェックを行うmain( )関数を定義します。

グラフィック画面の表示開始を行うstart( )関数，640×400モードにするall( )関数，画面消去を行うcls2( )関数を実行後，circle0( )関数を呼び出しています。円の中心座標(320, 200)，半径100の８分の１円弧を，青VRAMに描画します。

7 完全な円を描く

8分の1円弧が描けたので,これを8回行って完全な円にします。いまは描画方向を7とし描画開始座標(x＋r,y)としましたが，描画方向と描画開始座標を変えながら,あと7回描画する必要があります。描画方向/描画開始座標の関係は，表3-1のようになります。

描画方向	描画開始座標
0	(x－r,y)
1	(x,y－r)
2	(x,y＋r)
3	(x－r,y)
4	(x＋r,y)
5	(x,y＋r)
6	(x,y－r)
7	(x＋r,y)

この表3-1をよく見てください。描画開始座標は4種類しかありません。描画方向の0と3，1と6，2と5，4と7は同じ描画開始座標をもちます。この関係を利用すると，描画開始アドレスとドットアドレスの計算処理が4回で済みます。先の例で描画方向7の円弧を描きましたが，それに引き続いて描画方向4の円弧を描くことにすれば，CSRWコマンドのパラメータは，再度計算しなおす必要がありません。単に同じead/dadをパラメータ送出し，描画方向を4にセットしなおしてVECTWのコマンド/パラメータを送出するだけで描くことができます。つまり,

```
/* csrw */
send_command( 0x49 );
send_word_parameter( ead );
send_parameter( dad*16 );

/* vectw */
dir=4;  /* 描画方向を4に変更する */
send_command( 0x4c );
```

```
send_parameter( 0x20 + dir );
/* 他のパラメータは dir=7 の時と同じ */
send_word_parameter( dc_reg );
send_word_parameter( d_reg );
send_word_parameter( d2_reg );
send_word_parameter( d1_reg );
send_word_parameter( dm_reg );

/* vecte */
send_command( 0x6c );
```

この処理だけで，描画方向4の円弧も描けるわけです。このように，くふう次第で無駄な描画パラメータの計算が省けます。

VECTWコマンドでは，次のように重要なことがあります。描画方向7と描画方向4ではVECTWのパラメータの値は，dirを除いて変わりがありません。だからといって

```
dir=4;
send_command( 0x4c );    /* vectw */
send_parameter( 0x20 + dir );

send_command( 0x6c );    /* vecte */
```

のように，変更のないパラメータを送出せずに描画を行うこともできません。GDCは描画終了時に，描画パラメータレジスタDC，D，D2，D1，DMを初期値に設定してしまいます。DC，D，D2，D1，DMの初期値は

DC	D	D2	D1	DM
0	8	8	−1	−1

です。そのため描画がうまく行えないのです。注意してください。

VRAM1プレーンに円を描くcircle0( )関数は，今述べた描画方向0と3，1と6，2と5，4と7とを組み合わせる工夫をすると，リスト4のようになります。

◉リスト4

```
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2

void circle0( x_c,y_c,radius,plane,flag )
int x_c;                    /* x-coordinate */
int y_c;                    /* y-coordinate */
int radius;                 /* 半径 */
int plane;          /* 描画，消去するVRAMのプレーン番号 */
int flag;           /* 描画のモードを指定するフラグ */
{
        unsigned ead;
        unsigned ead_offset;
        unsigned dad;
        int xcrd;
        int ycrd;
        int dir;
        int dc_reg;     /* DC  レジスタ */
        int d_reg;      /* D   レジスタ */
        int d2_reg;     /* D2  レジスタ */
        int d1_reg;     /* D1  レジスタ */
        int dm_reg;     /* DM  レジスタ */


        /* textw */
        send_command( 0x78 );    /* textw command */
        send_word_parameter( 0xffff );  /* pattern */

        /* write */
        if( flag == PSET )
                send_command( 0x23 );
        if( flag == PRESET )
                send_command( 0x22 );
        if( flag == COMPLEMENT )
                send_command( 0x21 );

        /* vectw data */
        dc_reg = radius*100 /141+1;
        d_reg = radius-1;
        d2_reg = 2*d_reg;
        d1_reg = -1;
        dm_reg = 0;

/* dir 7 & 4 */
        xcrd = x_c + radius;
        ycrd = y_c;
```

```
        /* csrw */
        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*ycrd+xcrd / 16;
        dad = xcrd % 16;

        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        dir=7;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        dir=4;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );

/* dir 1 & 6 */
        xcrd = x_c;
        ycrd = y_c - radius;

        /* csrw */
        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*ycrd+xcrd / 16;
        dad = xcrd % 16;
```

```
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        dir=1;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        dir=6;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );

/* dir 3 & 0 */
        xcrd = x_c - radius;
        ycrd = y_c;

        /* csrw */
        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*ycrd+xcrd / 16;
        dad = xcrd % 16;

        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );
```

```
        /* vectw */
        dir=3;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        dir=0;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );

    /* dir 5 & 2 */
        xcrd = x_c;
        ycrd = y_c + radius;

        /* csrw */
        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*ycrd+xcrd / 16;
        dad = xcrd % 16;

        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        dir=5;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
```

```
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        dir=2;
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );
}
```

8回描画処理を行うので，少々長い定義となってしまいます。この関数はVRAM1プレーンに対してしか描画ができません(その代わり，ドット修正モードを任意に設定できるメリットもあります)。この関数を3回使ってはじめて通常の円がひとつ描けます。この節の目的である円の描画関数circle()は，リスト5のようになります。

◉リスト5

```
/*
        円を描く関数
*/
void circle(x,y,r,col)
unsigned x,y,r,col;
{
        if( (col & 1) != 0 )
                circle0(x,y,r,0,PSET);
        else
                circle0(x,y,r,0,PRESET);
        if( (col & 2) != 0 )
```

```
                    circle0(x,y,r,1,PSET);
        else
                    circle0(x,y,r,1,PRESET);
        if( (col & 4) != 0)
                    circle0(x,y,r,2,PSET);
        else
                    circle0(x,y,r,2,PRESET);
    }
```

circle関数は，第4引数で指定するcol(色コード)の値に従って円を描画するか，消去するかを変えながら，RGB3プレーンに対して描画/消去を繰り返して円を描いています。たとえば，colに3(紫)を選んだとします。紫＝青＋赤ですからVRAMのプレーン0，プレーン1に円をセットモードで描画し，プレーン2に円をクリアモードで描画します。つまり

```
    circle0(x,y,r,0,PSET);
    circle0(x,y,r,1,PSET);
    circle0(x,y,r,2,PRESET);
```

を実行します。上記のプログラムは，colのビットパターンを調べながらこれを実行しています。

円を描く関数circle()ができあがりました。これを試すメインプログラムをリスト6に示します。グラフィック画面を初期化し，画面の左から右に円の半径を増加させながら描画しています。

◉リスト6

```
/*
        circle 関数の使用例
*/
void main()
{
        int i;

        start();
        all();
        cls2();
        for( i=1; i<=150;i++)
                    circle(2*i+10,200,i,i % 7+1);
}
```

3 円弧描画

GDCは円弧を描画することもできます。この節では

```
void arc( x , y , r , col , start_angle , end_angle)
          int x   /* 中心X座標 */
          int y   /* 中心Y座標 */
          int r   /* 円弧の半径 */
          int col /* 色 */
          int start_angle /* 描画開始角＊10 */
          int end_angle   /* 描画終了角＊10 */
```

という円弧描画関数を定義することを目的としています。円弧描画は円描画と似ていますが，円よりも非常にやっかいな部分を含んでいます。まず，他の描画と同じようにVRAM1プレーンに対してのみ円弧描画を行う関数arc0( )を，定義することを目標にします。

1 円弧描画が困難なわけ

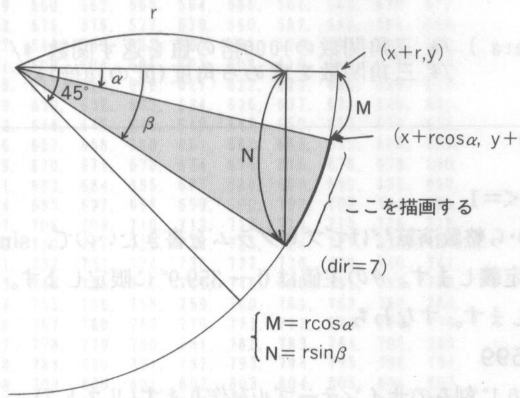
GDCの描画機能は，すべて0～45°の範囲でしか行えません。それ以上の角度にわたって描画を行うには，描画方向を変更しつつ描画を繰り返さなければなりません。簡単にするため，描画方向7の円弧描画を考えてみます。図3-8を見てください。GDCに描画開始中心角と描画終了中心角を直接指定できれば何の苦労もありません。しかし，実際にはそううまくはいきません。GDCに指定できるのは，描画開始アドレスから何ドット描画するか，そのうち何ドットを描画しないかです。全描画ドット数のことを

総描画ドット数　----------- N

描画しないドット数のことを

マスキングドット数　----- M

●図3-8●



と呼びます。まず問題なのはM，Nの計算です。描画開始中心角をα，描画終了中心角をβとすれば図3-8からわかるように

$$M = r * \sin(\alpha)$$

$$N = r * \sin(\beta)$$

という関係が成り立ちます。円弧を描画するという低レベルの処理を行うときに，三角関数の計算が必要なのは問題です。浮動小数点演算が必要になるため，描画前処理に時間がかかってしまいます。C言語にはsin関数が備わっているからよいものの，アセンブリ言語では浮動小数点演算ルーチンを作ったり，数値データプロセッサ(NDP)を装着してNDP用のプログラムを書くなどをしなければなりません。

演算精度が要求されない場合によく使われる方法に，テーブルサーチがあります。要するに，三角関数のテーブルをあらかじめ配列にもっておき，実行時にそれを参照する方法です。この方法ならば，C言語でもアセンブリ言語でも浮動小数点演算なしで高速に三角関数が計算できます。

2 sin関数の定義

まず，テーブルサーチを使ってsinを求める関数を定義しておきましょう。整数

値を返すのでisin()という名前にします。

```
int isin( theta ) /* 三角関数の1000倍の値を返す関数 */
int theta;        /* 三角関数を求める角度(度の10倍)*/
```

sin(θ)の値域は

$-1 <= \sin(\theta) <= 1$

です。計算速度の点から整数演算だけでプログラムを書きたいので，sin(θ)の1000倍の値を返す関数を定義します。θの主値は0～359.9°に限定します。θは角度の10倍を与えることにします。すなわち

$0 <= \theta <= 3599$

まず，0～90°までの0.1°刻みのサインテーブルを作ります(リスト7)。

◉リスト7

```
int sintable[]={
  0,   1,   3,   5,   6,   8,  10,  12,  13,  15,
 17,  19,  20,  22,  24,  26,  27,  29,  31,  33,
 34,  36,  38,  40,  41,  43,  45,  47,  48,  50,
 52,  54,  55,  57,  59,  61,  62,  64,  66,  68,
 69,  71,  73,  74,  76,  78,  80,  81,  83,  85,
 87,  88,  90,  92,  94,  95,  97,  99, 101, 102,
104, 106, 107, 109, 111, 113, 114, 116, 118, 120,
121, 123, 125, 127, 128, 130, 132, 133, 135, 137,
139, 140, 142, 144, 146, 147, 149, 151, 152, 154,
156, 158, 159, 161, 163, 165, 166, 168, 170, 171,
173, 175, 177, 178, 180, 182, 183, 185, 187, 189,
190, 192, 194, 195, 197, 199, 201, 202, 204, 206,
207, 209, 211, 213, 214, 216, 218, 219, 221, 223,
224, 226, 228, 230, 231, 233, 235, 236, 238, 240,
241, 243, 245, 246, 248, 250, 252, 253, 255, 257,
258, 260, 262, 263, 265, 267, 268, 270, 272, 273,
275, 277, 278, 280, 282, 284, 285, 287, 289, 290,
292, 294, 295, 297, 299, 300, 302, 304, 305, 307,
309, 310, 312, 313, 315, 317, 318, 320, 322, 323,
325, 327, 328, 330, 332, 333, 335, 337, 338, 340,
342, 343, 345, 346, 348, 350, 351, 353, 355, 356,
358, 359, 361, 363, 364, 366, 368, 369, 371, 372,
374, 376, 377, 379, 381, 382, 384, 385, 387, 389,
390, 392, 393, 395, 397, 398, 400, 401, 403, 405,
406, 408, 409, 411, 413, 414, 416, 417, 419, 421,
422, 424, 425, 427, 428, 430, 432, 433, 435, 436,
438, 439, 441, 443, 444, 446, 447, 449, 450, 452,
453, 455, 457, 458, 460, 461, 463, 464, 466, 467,
469, 471, 472, 474, 475, 477, 478, 480, 481, 483,
484, 486, 487, 489, 490, 492, 493, 495, 496, 498,
500, 501, 503, 504, 506, 507, 509, 510, 512, 513,
515, 516, 518, 519, 521, 522, 523, 525, 526, 528,
```

```
    529, 531, 532, 534, 535, 537, 538, 540, 541, 543,
    544, 546, 547, 549, 550, 551, 553, 554, 556, 557,
    559, 560, 562, 563, 564, 566, 567, 569, 570, 572,
    573, 575, 576, 577, 579, 580, 582, 583, 584, 586,
    587, 589, 590, 592, 593, 594, 596, 597, 599, 600,
    601, 603, 604, 605, 607, 608, 610, 611, 612, 614,
    615, 617, 618, 619, 621, 622, 623, 625, 626, 627,
    629, 630, 632, 633, 634, 636, 637, 638, 640, 641,
    642, 644, 645, 646, 648, 649, 650, 652, 653, 654,
    656, 657, 658, 660, 661, 662, 663, 665, 666, 667,
    669, 670, 671, 673, 674, 675, 676, 678, 679, 680,
    681, 683, 684, 685, 687, 688, 689, 690, 692, 693,
    694, 695, 697, 698, 699, 700, 702, 703, 704, 705,
    707, 708, 709, 710, 712, 713, 714, 715, 716, 718,
    719, 720, 721, 722, 724, 725, 726, 727, 728, 730,
    731, 732, 733, 734, 736, 737, 738, 739, 740, 741,
    743, 744, 745, 746, 747, 748, 750, 751, 752, 753,
    754, 755, 756, 758, 759, 760, 761, 762, 763, 764,
    766, 767, 768, 769, 770, 771, 772, 773, 774, 776,
    777, 778, 779, 780, 781, 782, 783, 784, 785, 786,
    788, 789, 790, 791, 792, 793, 794, 795, 796, 797,
    798, 799, 800, 801, 802, 803, 804, 805, 806, 807,
    809, 810, 811, 812, 813, 814, 815, 816, 817, 818,
    819, 820, 821, 822, 823, 824, 825, 826, 827, 828,
    829, 830, 830, 831, 832, 833, 834, 835, 836, 837,
    838, 839, 840, 841, 842, 843, 844, 845, 846, 847,
    848, 848, 849, 850, 851, 852, 853, 854, 855, 856,
    857, 858, 858, 859, 860, 861, 862, 863, 864, 865,
    866, 866, 867, 868, 869, 870, 871, 872, 872, 873,
    874, 875, 876, 877, 877, 878, 879, 880, 881, 882,
    882, 883, 884, 885, 886, 887, 887, 888, 889, 890,
    891, 891, 892, 893, 894, 894, 895, 896, 897, 898,
    898, 899, 900, 901, 901, 902, 903, 904, 904, 905,
    906, 907, 907, 908, 909, 909, 910, 911, 912, 912,
    913, 914, 914, 915, 916, 917, 917, 918, 919, 919,
    920, 921, 921, 922, 923, 923, 924, 925, 925, 926,
    927, 927, 928, 929, 929, 930, 931, 931, 932, 932,
    933, 934, 934, 935, 936, 936, 937, 937, 938, 939,
    939, 940, 940, 941, 942, 942, 943, 943, 944, 944,
    945, 946, 946, 947, 947, 948, 948, 949, 949, 950,
    951, 951, 952, 952, 953, 953, 954, 954, 955, 955,
    956, 956, 957, 957, 958, 958, 959, 959, 960, 960,
    961, 961, 962, 962, 963, 963, 964, 964, 965, 965,
    965, 966, 966, 967, 967, 968, 968, 969, 969, 969,
    970, 970, 971, 971, 971, 972, 972, 973, 973, 973,
    974, 974, 975, 975, 975, 976, 976, 977, 977, 977,
    978, 978, 978, 979, 979, 979, 980, 980, 980, 981,
    981, 981, 982, 982, 982, 983, 983, 983, 984, 984,
    984, 985, 985, 985, 985, 986, 986, 986, 987, 987,
    987, 987, 988, 988, 988, 989, 989, 989, 989, 990,
    990, 990, 990, 990, 991, 991, 991, 991, 992, 992,
    992, 992, 992, 993, 993, 993, 993, 993, 994, 994,
    994, 994, 994, 995, 995, 995, 995, 995, 995, 996,
    996, 996, 996, 996, 996, 996, 997, 997, 997, 997,
    997, 997, 997, 997, 998, 998, 998, 998, 998, 998,
    998, 998, 998, 998, 998, 999, 999, 999, 999, 999,
    999, 999, 999, 999, 999, 999, 999, 999, 999, 999,
    999, 999, 999, 999, 999, 999, 999, 999, 999, 999,
   1000
   };
```

これだけのテーブルを手で打つのは大変なので，このテーブルを作成するCプログラムを，リスト8に示します。

◉リスト8

```
/*
          sinの1000倍のテーブルを
          作成するプログラム
*/
#include <math.h>
#include <stdio.h>
#define PAI 3.141592654
void main()
{
          FILE *fp;
          double theta;
          int i;
          int s;

          fp = fopen("sintable.dat","w");
          fprintf(fp,"int sintable[]={¥n");
          for(i=0 ; i<=900 ; i++){
                    theta = i/10.0/180.0*PAI;
                    s =(int) ( 1000 * sin( theta ) );
                    fprintf(fp,"%4d",s);
                    if( i == 900)
                              fprintf(fp, "¥n");
                    else{
                              fprintf(fp,",");
                              if( (i % 10) == 9) fprintf(fp,"¥n");
                    }
          }
          fprintf(fp,"};");
          fclose( fp );
}
```

このプログラムを動作させるとsintable.datというファイルに，sinのテーブルを作成してくれます。

このテーブルは0～90°までのものです。ところでsin関数には，次の関係式が成り立ちます。

$$\sin(\theta)=\sin(\pi-\theta) \quad (1)$$

$$\sin(\theta)=-\sin(\theta-\pi) \quad (2)$$

$\sin(\theta) = -\sin(2\pi - \theta)$ (3)

この関係式を使えば，0～90°のsinの値があれば0～360°まで，すべてのsinの値が誘導できます。たとえば，91°のsinが必要ならば，(1)の関係式から

$\sin(91^\circ) = \sin(180^\circ - 91^\circ)$

となります。つまり

$\sin(91^\circ) = \sin(89^\circ)$

となって，90°以下のsinの値を計算すれば済むことがわかります。190°のsinならば(2)の関係式から

$\sin(190^\circ) = -\sin(190^\circ - 180^\circ) = -\sin(10^\circ)$

となり，10°のsinを求めて符号を反転させればよいことがわかります。以上のことから，1000倍のsinの値を返す関数isin()は，次のように定義できます。

◉リスト9

```
int isin( theta ) /* 三角関数の1000倍の値を返す関数 */
int theta;        /* 三角関数を求める角度(度の10倍)*/
{
        if( theta <= 900 )
                return( sintable[ theta ] );
        if( (900 < theta) && (theta <= 1800) )
                return( sintable[ 1800-theta ] );
        if( (1800 < theta) && (theta <= 2700) )
                return( -sintable[ theta-1800 ] );
        if( 2700 < theta )
                return( -sintable[ 3600-theta ] );
}
```

sinを求める準備ができたので，円弧描画を考えていきたいと思います。描画方向7で中心座標(x, y)，半径r，描画プレーン番号plane，ドット修正モードflag，描画開始角度alpha，描画終了角度betaであるような円弧を描く関数arc07を考えましょう。すなわち

```
void arc07( x , y , r, plane , flag , alpha , beta )
        int x;              /* 中心X座標 */
        int y;              /* 中心Y座標 */
        int r;              /* 円弧の半径 */
        int plane;          /* 描画プレーンの番号 */
        int flag;           /* ドット修正モード */
        int alpha;          /* 描画開始角(度の10倍) */
        int beta;           /* 描画終了角(度の10倍) */
```

円弧描画を行わせるためには，次のコマンド送出が必要です。

- TEXTW コマンド
- WRITE コマンド
- CSRW コマンド
- VECTW コマンド
- VECTE コマンド

3 TEXTW コマンド

円弧を描くラインパターンの設定です。円弧では，最大8個の円弧を組み合わせて描画するため，一点鎖線などを指定してもうまく模様が合わないこともあると思います。実用上は実線や破線を使うことになるでしょう。ここでも，実線の指定をします。送出するコマンドとパラメータは，次のとおりです。

コマンド	0AAH
パラメータ	0FFH
	0FFH

4 WRITE コマンド

ドット修正モードの指定を行います。円弧描画で注意すべき点は，マスキング区

間での動作です。マスキング区間というのは，総描画ドット数のうち描画をしない区間です。この間も，GDCは描画アドレスの更新を行い続けています。つまりEADやdADはどんどん変化しているわけです。しかし，ドット修正動作はしていません。つまりマスキング区間では(たとえ，どのようなドット修正モードが指定されていても)VRAMのデータは変化しません。

円弧描画でも指定できるドット修正モードは同じです。すなわちSET，CLEAR，COMPLEMENT，REPLACEが使用できます。WRITEで送出すべきコマンドは，次のとおりです。

コマンド	20H 21H 22H 23H	REPLACEモード時 COMPLEMENTモード時 CLEARモード時 SETモード時
パラメータ	なし	

これらTEXTW/WRITEコマンドは描画ルーチンの先頭で一度だけ送出します。

5 CSRWコマンド

円弧描画を開始するVRAMのアドレスをCSRWコマンドでGDCに指示します。いま，描画方向7で考えていますが，図3-8のような円弧を描く場合，GDCに指示すべき座標は

(x+r＊cos(α), y+r＊sin(α))

ではなく単に

(x+r, y)

です。VRAMに描画を開始する座標は，確かに

(x+r＊cos(α), y+r＊sin(β))

なのですがGDCに指示するのは，(x+r, y)のアドレスです。GDCは(x+r, y)というドットから，マスキングドット数だけ描画(ドット修正動作)をせずEAD/dADの値だけを更新していきます。Mドット分が終わったならば，総描画ドット数Nになるまで描画(ドット修正動作)を行います。

ですから，点(x+r, y)のGDCから見た描画アドレス/ドットアドレスを計算す

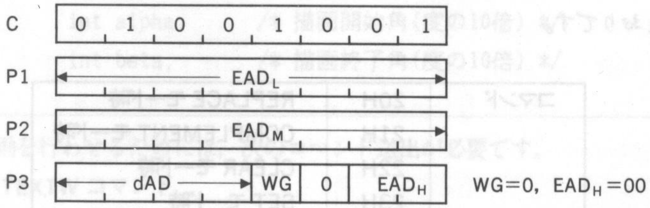
る描画前処理が必要です。

描画開始アドレスEADとドットアドレスdAD，および(x, y)，rの関係は次のとおりです(プレーン0に描画する場合)。

ead＝0x4000＋40 * y＋(x＋r)/16;

dad＝(x＋r) % 16;

CSRWで送出するコマンド/パラメータは，次のとおりです。



6 VECTWコマンド

円弧描画でのVECTWコマンドの役割は，円弧の半径，円弧の描画方向，描画開始角，描画終了角をGDCに教えることです。描画開始角，描画終了角のところで前出のsin関数が必要となります。描画前処理としてはVECTWコマンドが一番多くの手間がかかります。

VECTWで送出するコマンド/パラメータは次ページ上のとおりです。

設定するパラメータの値は，次のとおりです。

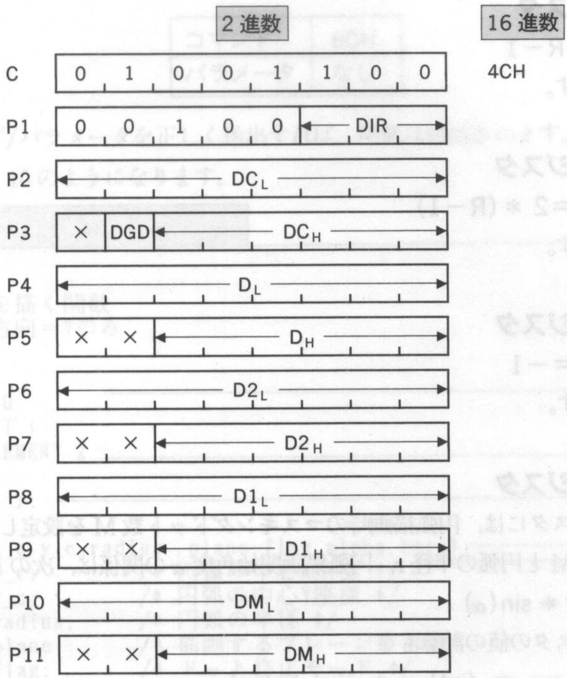
■ DIR

円弧を描画する方向です。描画方向は円の場合と同じです。

■ DCレジスタ

DCレジスタには総描画ドット数Nを与えます。先ほど説明したように，総描画ドット数Nは円弧の半径をr，円弧の描画終了角をβとすれば

N＝r×sin(β)



という式で計算できます。こういうパラメータはユーザが指定するのではなく，描画プロセッサの内部で行ってもらいたいものです。

ちなみに，円描画の場合は

DC＝(r/sqr(2)を切り上げた値)

を指定していましたが，これは円弧描画で $\beta=45^\circ$ としたのと同じです。

1/sqr(2)＝sin(45°)

つまり円描画は，円弧描画の特殊なケースなわけです。

DC レジスタの値をCでプログラムを組む場合は

```
dc_reg = (int) r * sin( beta );
```

とすると時間がかかるので，前出の isin 関数を使って

```
dc_reg = ( int )(long)r *(long)isin( beta )/1000L;
```

のように整数演算だけで計算します。

■ Dレジスタ

D＝R－1

を設定します。

■ D2レジスタ

D2＝2＊(R－1)

を設定します。

■ D1 レジスタ

D1＝－1

を設定します。

■ DMレジスタ

DMレジスタには，円弧描画時のマスキングドット数Mを設定します。マスキングドット数Mと円弧の半径r，円弧描画開始角度αの関係は，次のとおりです。

M＝r＊sin(α)

DMレジスタの値の計算も

```
dm_reg = (int) r * sin( beta );
```

とするのではなく，先ほど定義したisin()関数を使って

```
dm_reg = ( int )(long)r *(long)isin( alpha )/1000L;
```

で行います。

円描画のときは，DMレジスタの値は

```
DM   =  0
```

と設定していました。これは円弧描画の$\alpha=0°$のときと同じ値です。

R＊sin(0)＝0

円の場合はすべての点を描かなければならないので，マスキングドット数Mは0だったわけです。

7 VECTE コマンド

GDCに円弧を描き始めることを命令します。VECTEコマンドは

コマンド	6CH
パラメータ	なし

以上のコマンド/パラメータを正しく送出すれば，円弧は描画されます。よって求める関数はリスト10のようになります。

◉リスト10

```
/*
      円弧を描く関数
      描画方向＝7のみ
*/

#define PSET 0                                        ❶
#define PRESET 1
#define COMPLEMENT 2

int isin( int );                                      ❷

void arc07( x_c,y_c,radius ,plane,flag,alpha,beta)    ❸
      int x_c;           /* 円弧の中心X座標 */        ❹
      int y_c;           /* 円弧の中心Y座標 */
      int radius;        /* 円弧の半径 */
      int plane;         /* 描画するプレーン番号 */
      int flag;          /* ドット修正モード */
      int alpha;         /* 描画開始角(度の10倍)*/
      int beta;          /* 描画開始角(度の10倍)*/
{
      unsigned ead;                                   ❺
      unsigned ead_offset;
      unsigned dad;
      int xcrd;
      int ycrd;
      int dir;
      int dc_reg;        /* DC  レジスタ */
      int d_reg;         /* D   レジスタ */
      int d2_reg;        /* D2  レジスタ */
      int d1_reg;        /* D1  レジスタ */
      int dm_reg;        /* DM  レジスタ */

      /* textw */                                     ❻
      send_command( 0x78 );   /* textw command */
      send_word_parameter( 0xffff );  /* pattern */
```

```
    /* write */
    if( flag == PSET )
            send_command( 0x23 );
    if( flag == PRESET )                                        ❼
            send_command( 0x22 );
    if( flag == COMPLEMENT )
            send_command( 0x21 );

    /* vectw data */                                            ❽
    dc_reg = ( int )(long)radius*(long)isin( beta )/1000L;

    d_reg = radius-1;
    d2_reg = 2*d_reg;                                           ❾
    d1_reg = -1;
    dm_reg = ( int )(long)radius*(long)isin( alpha )/1000L;     ❿

/* dir 7 */
    xcrd = x_c + radius;                                        ⓫
    ycrd = y_c;
    /* csrw */
    ead_offset=0x4000U+0x4000U*plane;
    ead = ead_offset+40*ycrd+xcrd / 16;                         ⓬
    dad = xcrd % 16;

    send_command( 0x49 );
    send_word_parameter( ead );                                 ⓭
    send_parameter( dad*16 );

    /* vectw */
    dir=7;                                                      ⓮
    send_command( 0x4c );                                       ⓯
    send_parameter( 0x20 + dir );                               ⓰
    send_word_parameter( dc_reg );
    send_word_parameter( d_reg );
    send_word_parameter( d2_reg );                              ⓱
    send_word_parameter( d1_reg );
    send_word_parameter( dm_reg );

    /* vecte */
    send_command( 0x6c );                                       ⓲
}
```

解説

❶必要な定数をマクロ定義します。

❷先ほど解説したsinを返す関数を，このファイル中でも使用できるように宣言します。

❸描画方向7の円弧を描く関数の定義です。値は返さないのでvoid型で宣言します。

❹仮引数の宣言です。

❺この関数の中で使用する変数を宣言します。

❻まず，TEXTコマンドを送出して円弧の描画パターンを指定します。TEXTWコマンドは78H，パラメータは実線なのでFFFFHを送出しています。

❼ドット修正モードを指定するためWRITEコマンドを送出します。円弧を描くか，消去するかVRAMのデータを反転させるかなどに応じて送出するコマンドを23H，22H，21Hから選んでいます。

❽描画パラメータレジスタの値を計算します。DCレジスタの値も決めます。isin( )関数は1000倍のsinの値を返します。つまり

　−1000＜＝isin(beta)＜＝1000

です。一方，円弧の半径radiusは200程度になることがあります。整数演算だとradiusとisin(beta)の乗算を行うときにオーバーフローを起こしてしまいます。それを避けるために(long)を使って値を変換しています。

❾D，D2，D1の値を設定します。

❿DMレジスタの値もやっかいです。DCレジスタのときと同様な方法で設定しています。(long)を使ってオーバーフローしないようにします。

⓫描画開始座標(xcrd，ycrd)を計算します。いま描画方向が7なので(x＋r，y)となります。

⓬いま計算した描画開始座標から描画開始アドレスead，描画開始ドットアドレスdadを計算します。描画するプレーンに応じてアドレスにオフセットを加えています。

⓭CSRWのコマンドとパラメータを送出します。CSRWコマンドは49Hです。

⓮VECTWコマンドとパラメータを送出します。描画方向は7です。

⓯VECTWコマンド4CHをコマンド送出します。

⓰描画方向を送出します。

⓱各描画パラメータレジスタの値を送出します。

⓲ VECTE コマンド(6CH)を送って，やっと円弧が描画できます。

これで描画方向7についての円弧描画ルーチンができました。ほかの描画方向についても同じようにして行えます。試しにこの関数を呼んでみましょう。

```
/*
        arc07()関数のチェック
*/
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2
void main()
{
        start();
        all();
        cls2();

        arc07( 320 , 200 , 200 , 0 , PSET , 100 , 250 );
}
```

中心(320, 200)，半径200，描画開始角10°，描画終了角25°の円弧をプレーン0に描画します。

さて，描画方向によって変えなければならないのは，描画開始座標とVECTWコマンドのパラメータDIRです。描画方向7に対する円弧描画は

```
void arc07(x_c, y_c, radius , plane, flag, alpha, beta)
```

と定義していましたが，これに描画方向の仮引数を付加した

```
void arc00(x_c, y_c, radius , plane, flag, alpha, beta, dir)
```

も容易に作成できます。大きな変更点は，描画開始座標を

```
xcrd = x_c + radius;
ycrd = y_c;
```

としていた部分を

```
if( (dir == 7) || (dir == 4) ){
        xcrd = x_c + radius;
        ycrd = y_c;
}
if( (dir == 0) || (dir == 3) ){
        xcrd = x_c - radius;
        ycrd = y_c;
}
if( (dir == 1) || (dir == 6) ){
        xcrd = x_c;
        ycrd = y_c - radius;
}
if( (dir == 2) || (dir == 5) ){
        xcrd = x_c;
        ycrd = y_c + radius;
}
```

のように描画方向によって計算方法を変える点です。ところで，TEXTWコマンドとWRITEコマンドは毎回送出するのではなく，描画の先頭で1回呼べば済みます。そこで，arc00()関数ではTEXTWコマンドとWRITEコマンドの送出をメインルーチンのほうにゆだねています。arc00()関数の定義はリスト11のようになります。

◉リスト11

```
/*
        円弧を描く関数
        描画方向を引数で指定
*/

#define PSET 0
#define PRESET 1
#define COMPLEMENT 2

int isin( int );        /* sinの1000倍を返す関数 */

void arc00( x_c,y_c,radius ,plane,flag,alpha,beta,dir)
        int x_c;            /* 円弧の中心X座標 */
        int y_c;            /* 円弧の中心Y座標 */
        int radius;         /* 円弧の半径 */
        int plane;          /* 描画するプレーン番号 */
        int flag;           /* ドット修正モード */
        int alpha;          /* 描画開始角(度の10倍)*/
        int beta;           /* 描画開始角(度の10倍)*/
        int dir;            /* 描画方向 */
{
        unsigned ead;   /* 描画開始アドレス */
```

```
        unsigned ead_offset;
        unsigned dad;       /* 描画開始ドットアドレス */
        int xcrd;           /* 描画開始X座標 */
        int ycrd;           /* 描画開始Y座標 */
        int dc_reg;         /* DC  レジスタ */
        int d_reg;          /* D   レジスタ */
        int d2_reg;         /* D2  レジスタ */
        int d1_reg;         /* D1  レジスタ */
        int dm_reg;         /* DM  レジスタ */

        /* vectw data */
        dc_reg = ( int )(long)radius*(long)isin( beta )/1000L;

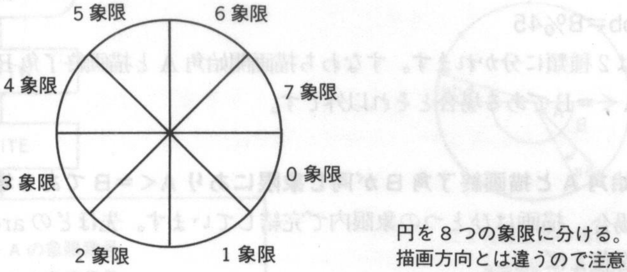
        d_reg = radius-1;
        d2_reg = 2*d_reg;
        d1_reg = -1;
        dm_reg = ( int )(long)radius*(long)isin( alpha )/1000L;
        if( (dir == 7) || (dir == 4) ){
                xcrd = x_c + radius;
                ycrd = y_c;
        }
        if( (dir == 0) || (dir == 3) ){
                xcrd = x_c - radius;
                ycrd = y_c;
        }
        if( (dir == 1) || (dir == 6) ){
                xcrd = x_c;
                ycrd = y_c - radius;
        }
        if( (dir == 2) || (dir == 5) ){
                xcrd = x_c;
                ycrd = y_c + radius;
        }
        /* csrw */
        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*ycrd+xcrd / 16;
        dad = xcrd % 16;

        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( dad*16 );

        /* vectw */
        send_command( 0x4c );
        send_parameter( 0x20 + dir );
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );
}
```

●図 3-9 ●



さて，この関数を使って任意の方向の円弧を描けるようにします。図 3-9 を見てください。45°ごとに 0，1，2，…，7 というように象限に分けます(これは GDC の描画方向番号とは違います)。ある象限に対して，時計回りに角度 A から角度 B までを描く関数 arc00 _internal( )を作成します。この関数は

```
void arc0_internal(x,y,r,plane,flag,a,b,i)
int x;  /* 円弧のX中心座標 */
int y;  /* 円弧のY中心座標 */
int r;  /* 円弧の半径 */
int plane;      /* 描画プレーン番号 */
int flag;       /* ドット修正モード */
int a;          /* 象限内での描画開始角 */
int b;          /* 象限内での描画終了角 */
int i;          /* 象限番号 */
```

と定義されているものとします。この関数を利用して，全周にわたって円弧が描けるようにします。円弧の描画開始角を A，描画終了角を B とし，A，B の象限番号を Ia，Ib とすれば Ia，Ib は次の計算式で求めることができます。

Ia＝A/45

Ib＝B/45

各象限内での描画開始角aa，描画終了角bbは，次のとおりです。

aa＝A%45

bb＝B%45

描画は2種類に分かれます。すなわち描画開始角Aと描画終了角Bが同じ象限にあり，A＜＝Bである場合とそれ以外です。

描画開始角Aと描画終了角Bが同じ象限にありA＜＝Bである場合

この場合，描画はひとつの象限内で完結しています。先ほどのarc00_internal()関数を使えば

arc0_internal(x, y, r, plane, flag, aa, bb, ia);

で描画できます。ひとつの象限内で角度aaからbbまで描けば終わりです。

それ以外の場合

この場合は複数の象限にまたがって描画する必要があります。まず，Aの含まれる象限Iaの中では，aaから45°までを描きます。arc00_internal()関数を使えば

arc0_internal(x, y, r, plane, flag, aa, 450, ia);

となります。角度が10倍されていることに注意してください。

次に，Bの含まれる象限のひとつ手前まで円弧を連続的に描きます。AからBの間にある象限の個数をNとすれば

N＝Ib－Ia－1

です。A＞Bという場合も考えると

N＝(Ib－Ia－1＋8)%8

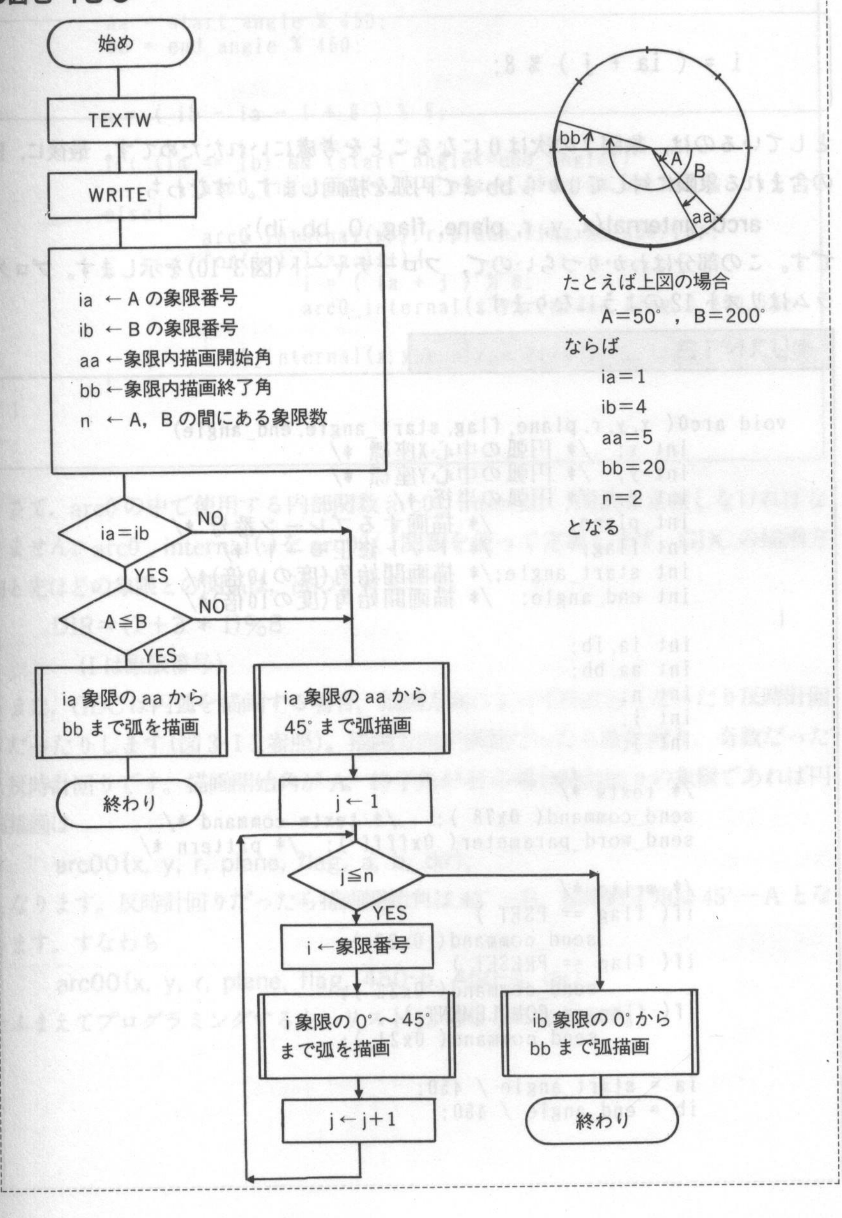
という計算式で計算するのが一般的です。

象限Ia＋1，Ia＋2，…，Ib－1に対して0～45°までの円弧を描きます。ループ変数をj描画する象限番号をiに入れるとすれば，次のようになります。

```
for(j=1;j<=n;j++){
        i = ( ia + j ) % 8;
        arc0_internal(x,y,r,plane,flag,0,450,i);

}
```

●図3-10●



始め
TEXTW
WRITE
ia ← A の象限番号
ib ← B の象限番号
aa ←象限内描画開始角
bb ←象限内描画終了角
n ← A，B の間にある象限数
ia＝ib
NO
YES
A≦B
NO
YES
ia 象限の aa から
bb まで弧を描画
終わり
ia 象限の aa から
45° まで弧描画
j ← 1
j≦n
YES
i ←象限番号
i 象限の 0°～45°
まで弧を描画
j ← j＋1
ib 象限の 0° から
bb まで弧描画
終わり
bb
A
B
aa
たとえば上図の場合
A＝50°，B＝200°
ならば
ia＝1
ib＝4
aa＝5
bb＝20
n＝2
となる

象限番号 i を求めるときに

```
i = ( ia + j ) % 8;
```

としているのは，象限 7 の次は 0 になることを考慮にいれたためです。最後に，B の含まれる象限に対して 0 から bb まで円弧を描画します。すなわち

arc0_internal(x, y, r, plane, flag, 0, bb, ib);

です。この部分はわかりづらいので，フローチャート(図 3-10)を示します。プログラムはリスト 12 のようになります。

◉リスト 12

```
void arc0( x,y,r,plane,flag,start_angle,end_angle)
        int x;  /* 円弧の中心X座標 */
        int y;  /* 円弧の中心Y座標 */
        int r;  /* 円弧の半径 */
        int plane;        /* 描画するプレーン番号 */
        int flag;         /* ドット修正モード */
        int start_angle;/* 描画開始角(度の10倍)*/
        int end_angle;  /* 描画開始角(度の10倍)*/
{
        int ia,ib;
        int aa,bb;
        int n;
        int i;
        int j;

        /* textw */
        send_command( 0x78 );   /* textw command */
        send_word_parameter( 0xffff );  /* pattern */

        /* write */
        if( flag == PSET )
                send_command( 0x23 );
        if( flag == PRESET )
                send_command( 0x22 );
        if( flag == COMPLEMENT )
                send_command( 0x21 );

        ia = start_angle / 450;
        ib = end_angle / 450;
```

```
    aa = start_angle % 450;
    bb = end_angle % 450;

    n = ( ib - ia - 1 + 8 ) % 8;

    if( (ia == ib) && (start_angle<=end_angle))
            arc0_internal(x,y,r,plane,flag,aa,bb,ia);
    else{
            arc0_internal(x,y,r,plane,flag,aa,450,ia);
            for(j=1;j<=n;j++){
                    i= ( ia + j ) % 8;
                    arc0_internal(x,y,r,plane,flag,0,450,i);
            }
            arc0_internal(x,y,r,plane,flag,0,bb,ib);
    }
}
```

さて，arc0の中で使用する内部関数arc0_internal( )関数を定義しなければなりません。arc0_internal( )をarc00( )関数を使って定義します。GDCの描画方向と先ほどの象限との関係は，次のとおりです。

DIR＝(7＋3 ＊ I)%8

(Iは象限番号)

また，GDCは円弧を描画する場合，描画方向によって時計回りだったり反時計回りだったりします(図3-11参照)。描画方向が偶数だったら時計回り，奇数だったら反時計回りです。描画開始角がA，終了角がBの場合時計回りの象限であれば円弧描画は

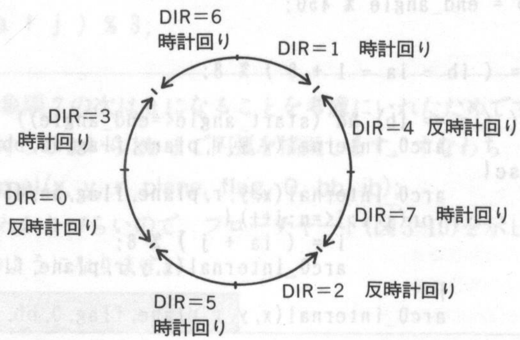
arc00(x, y, r, plane, flag, a, b, dir);

となります。反時計回りだったら描画開始角は45°－B，描画終了角は45°－Aとなります。すなわち

arc00(x, y, r, plane, flag, 450-b, 450-a, dir);

をふまえてプログラミングすると，リスト13のようになります。

●図 3-11 ●



◉リスト 13

```
void arc0_internal(x,y,r,plane,flag,a,b,i)
int x,y,r,plane,flag,a,b,i;
{
        int dir;

        dir = ( 7 + 3 * i ) % 8;
        if( (i%2) == 0)
                arc00(x,y,r,plane,flag,a,b,dir);
        else
                arc00(x,y,r,plane,flag,450-b,450-a,dir);
}
```

いま定義したのは，ひとつのプレーンに対してのみ描画する関数です。これを描画したい色コードに応じてPSET/PRESETを変えながらRGBを3回描画して,ようやく完全な円弧描画関数(リスト14)ができあがります。

◉リスト 14

```
/*
        円弧を描画する関数
*/

void arc( x,y,r,col,start_angle,end_angle)
        int x;  /* 円弧の中心X座標 */
        int y;  /* 円弧の中心Y座標 */
        int r;  /* 円弧の半径 */
```

```
        int col;          /* 色 */
        int start_angle;/* 描画開始角
        (度の10倍  0以上3599以下)*/
        int end_angle;  /* 描画開始角
        (度の10倍  0以上3599以下)*/
{
        if( (col & 1) != 0 )
            arc0( x,y,r,0,PSET,start_angle,end_angle);
        else
            arc0( x,y,r,0,PRESET,start_angle,end_angle);
        if( (col & 2) != 0 )
            arc0( x,y,r,1,PSET,start_angle,end_angle);
        else
            arc0( x,y,r,1,PRESET,start_angle,end_angle);
        if( (col & 4) != 0 )
            arc0( x,y,r,2,PSET,start_angle,end_angle);
        else
            arc0( x,y,r,2,PRESET,start_angle,end_angle);
}
```

このように円弧描画はかなり面倒です。いま定義したarc()関数を試すデモプログラムを次に示します。任意の大きさの円弧を任意な中心角，任意の色で1000個描画するものです。

◉リスト15

```
/*
        円弧を描く関数のテスト
*/
#include <stdlib.h>
void main()
{
        int i;
        int r,c,s,e;

        for(i=1;i<=1000;i++){
                r=rand() % 198;
                c=rand() % 7+1;
                s=rand() % 3600;
                e=rand() % 3600;
                arc(320,200,r,c,s,e);
        }
}
```

4 扇形描画

直線描画と円弧描画を利用すれば，扇形描画も行えます。この節では

```
void arc2( x , y , r , col , start_angle , end_angle)
     int x   /* 中心X座標 */
     int y   /* 中心Y座標 */
     int r   /* 扇形の半径 */
     int col /* 色 */
     int start_angle /* 描画開始角＊10 */
     int end_angle   /* 描画終了角＊10 */
```

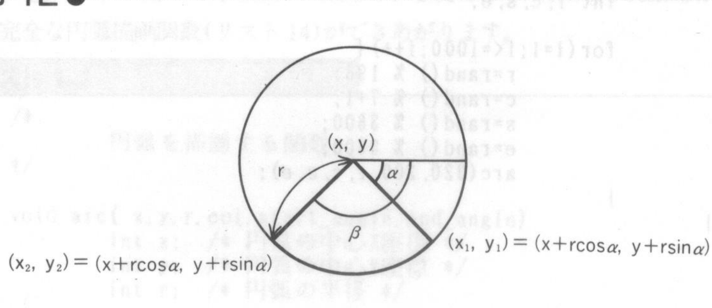
という関数を定義することを目的とします。

問題になるのは中心点から円弧へ直線を結ぶところでしょう。円弧描画のときに指定するのは中心座標だけだからです。円周上の点の座標の計算が必要です。図3-12からもわかると思いますが，円周上の端点の座標を(x_1, y_1) (x_2, y_2)，中心座標を(x, y)とすれば

$$x_1 = x + r * \cos(\alpha)$$
$$y_1 = y + r * \sin(\alpha)$$

$$x_2 = x + r * \cos(\beta)$$
$$y_2 = y + r * \sin(\beta)$$

●図3-12●



となります。sin関数は円弧描画の節で定義しましたが，cos関数は定義していません。しかし

$\cos(\theta) = \sin(\pi/2 - \theta)$

の関係が成り立つので，sin関数からcos関数を誘導することができます。プログラムはリスト15のようになります。

◉リスト15

```
/*
   cosの1000倍を返す関数
*/
int icos( theta )
int theta;
{
        int temp;
        temp=900-theta;
        if( temp<0 ) temp += 3600;
        return( isin( temp ) );
}
```

isin関数の引数が0から3599だったので

```
temp=900-theta;
if( temp<0 ) temp += 3600;
```

ように900−thetaが負の場合，3600を加えています。icos関数があれば，扇形を描画する関数は次のように定義できます(リスト16)。

◉リスト16

```
void arc2( x,y,r,col,start_angle,end_angle)
        int x;  /* 円弧の中心X座標 */
        int y;  /* 円弧の中心Y座標 */
        int r;  /* 円弧の半径 */
        int col;         /* 色 */
        int start_angle;/* 描画開始角
        (度の10倍  0以上3599以下)*/
        int end_angle;  /* 描画開始角
        (度の10倍  0以上3599以下)*/
{
```

```
        int  x1,y1;
        int  x2,y2;

        arc(x,y,r,col,start_angle,end_angle); ──────────❶
        x1=(int)(x + (long)r*icos( start_angle )/1000L );─┐❷
        y1=(int)(y + (long)r*isin( start_angle )/1000L );─┘
        line(x,y,x1,y1,col); ─────────────────────────❸
        x2=(int)( x + (long)r*icos( end_angle )/1000L ); ─┐❹
        y2=(int)( y + (long)r*isin( end_angle )/1000L ); ─┘
        line(x,y,x2,y2,col); ─────────────────────────❺
}
```

解説

❶まず，arc関数を使って円周部を描画します。

❷円周の描画開始点の座標(x_1, y_1)を計算します。

❸rとicos()，rとisin()の乗算時にオーバーフローすることが考えられるのでロングに変換して乗算しています。

そして(x, y)(x_1, y_1)を直線で結びます。

❹同様に円周の描画終了点の座標(x_2, y_2)を計算します。

❺そして，(x, y)(x_2, y_2)を直線で結びます。

計算誤差で直線と円周部との接続が1ドット程度ずれることがあります。半径が大きい場合には，それが顕著になります。もしそれを避けたければ，三角関数のテーブルの値を10000倍にしたり，実数演算にするなどの方法をとる必要があります。実用上は，この程度の計算精度でよいと思います。

arc2関数をテストするプログラムをリスト17に示します。

●リスト 17

```
/*
        arc2関数のテスト
*/
#include <stdlib.h>
#define TRUE 0
#define FALSE -1
void main()
{
        int x;
        int ox=0;
        int angle=450;
        int oangle=450;
        int dangle=-100;
        int first = TRUE;


        start();
        all();
        cls2();
        for(x=50;x<=600;x+=5){
                if( first == FALSE ){
                        vsync();
                        arc2(ox,200,15,0,oangle,3600-oangle);
                }
                else
                        first = FALSE;
                arc2(x,200,15,1,angle,3600-angle);
                ox=x;
                oangle=angle;

                angle=angle+dangle;
                if( (angle>450) || (angle<150)) dangle = -dangle;
        }
}
```

5

四辺形描画

四辺形描画とは，長方形を描くことをいいます。GDCは辺が水平な長方形と45°に傾いた長方形を描くことができます。この節では，辺が水平なものを描くことにします(図3-13)。

次の関数を定義することを目標にします。

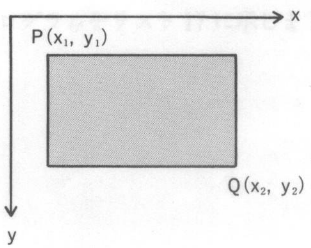
```
void box( x1 ; y1 , x2 , y2 , color )
int x1;          /*     点PのX座標     */
int y1;          /*     点PのY座標     */
int x2;          /*     点QのX座標     */
int y2;          /*     点QのY座標     */
int color;       /*     色             */
```

四辺形描画は，ほかの描画に比べて送出するパラメータが簡単です。直線ルーチンを4回呼べば四辺形描画になりますが，それでは送出するパラメータが多くなり，時間がかかってしまいます。

四辺形描画を行うためには，次のコマンドをGDCに送出します。

- TEXTW コマンド
- WRITE コマンド

●図3-13●



- CSRW コマンド
- VECTW コマンド
- VECTE コマンド

1 TEXTW コマンド

四辺形を描くラインパターンの設定です。四辺形描画は連続的に4つの辺を描くので，複雑なラインパターンを設定しても頂点の部分で不連続になったりはしません。実線で四辺形を描画するときは，送出するコマンドとパラメータが次のようになります。

コマンド	0AAH
パラメータ1	0FFH
パラメータ2	0FFH

2 WRITE コマンド

ドット修正モードの指定を行います。SET，CLEAR，COMPLEMENT，REPLACEの各モードを指定します。WRITEで送出すべきコマンドは，以下のとおりです。

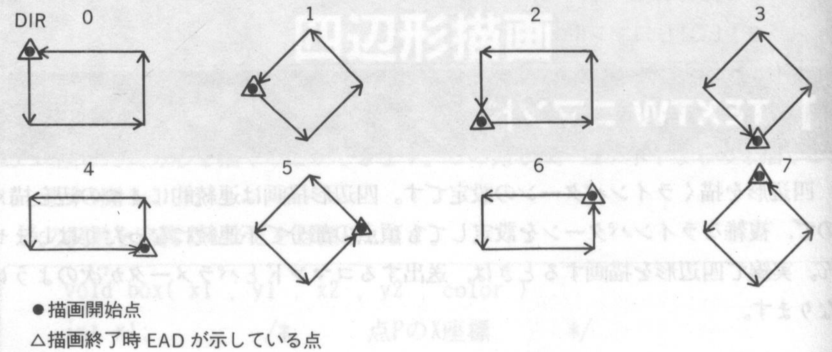
コマンド	20H　REPLACE モード時 21H　COMPLEMENT モード時 22H　CLEAR モード時 23H　SET モード時
パラメータ	なし

TEXTW，WRITE コマンドは，描画前に一度送出します。

3 CSRW コマンド

四辺形の描画開始アドレスをGDCに知らせます。四辺形描画でおもしろいのは，

●図3-14●

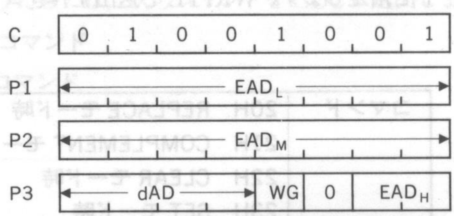


EAD/dADレジスタの値です。図3-14に示すように，GDCは描画を終了すると，グルッと一周してもとのところに戻っています。

描画を開始する点の座標を(x_1, y_1)とすれば

```
ead = 0x4000 + 40 * y1 +  x1/16;
dad = x1 % 16;
```

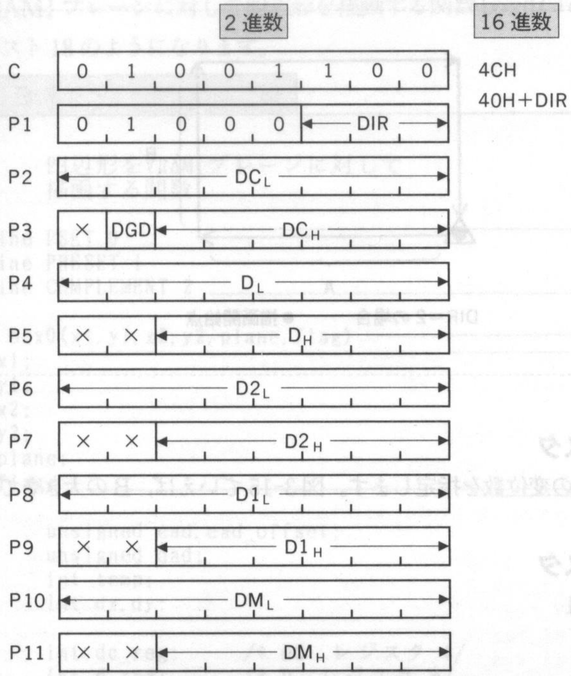
です。CSRWで送出するコマンド/パラメータは，次のようになります。



4 VECTWコマンド

描画用パラメータレジスタの値を設定します。ほかの描画に比べて設定するパラメータレジスタの値はシンプルです。

送出するコマンド/パラメータは，次のとおりです。



設定するパラメータの値は，次のとおりです。

■ DIR

四辺形を描画する方向です(図3-14)。

このように，描画方向DIRを1，3，5，7にセットすることで45°傾いた四辺形を描画できます。

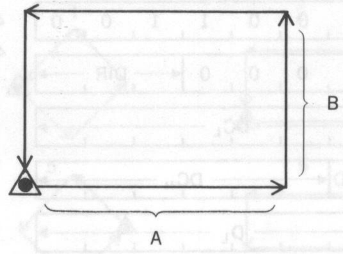
■ DCレジスタ

3を設定します。

■ Dレジスタ

第1描画方向のドット数を指定します。図3-15でいえばAの大きさに相当します。

●図 3-15●



■ D2 レジスタ

第 2 描画方向の変位数を指定します。図 3-15 でいえば，B の大きさです。

■ D1 レジスタ

D1＝－1

を設定します。

■ DM レジスタ

D レジスタに設定したのと同じ値をセットします。

5 VECTE コマンド

四辺形描画を開始させるために，VECTE コマンドを送出します。円描画のときと違い，四辺形描画では 4 辺を連続して一度に描画します。

コマンド	6CH
パラメータ	なし

四辺形描画は，8 方向いずれにも描画することができますが，この節で扱う水平な四辺形の場合，左上の座標と右下の座標を操作することで描画方向 DIR＝0 だけですべてのプログラムを作ることができます。

まず，VRAM1プレーンに対して四辺形を描画する関数box0()を定義します。box0()はリスト18のようになります。

●リスト 18

```
/*
      四辺形をVRAM1プレーンに対して
      描画する関数
*/
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2

void box0(x1,y1,x2,y2,plane,flag)
int x1;
int y1;
int x2;
int y2;
int plane;
int flag;
{
      unsigned ead,ead_offset;
      unsigned dad;
      int temp;
      int dx,dy;

      int dc_reg;       /* DC  レジスタ */
      int d_reg;        /* D   レジスタ */
      int d2_reg;       /* D2  レジスタ */
      int d1_reg;       /* D1  レジスタ */
      int dm_reg;       /* DM  レジスタ */

      /* textw */
      send_command( 0x78 );    /* textw command */
      send_word_parameter( 0xffff ); /* pattern */

      /* write */
      if( flag == PSET )
            send_command( 0x23 );
      if( flag == PRESET )
            send_command( 0x22 );
      if( flag == COMPLEMENT )
            send_command( 0x21 );
```

```
        if( x1 > x2 ){
                temp=x2; x2=x1; x1=temp;                ❼
        }
        if( y1 > y2 ){
                temp=y2; y2=y1; y1=temp;                ❽
        }
        dx = x2 - x1;                                   ❾
        dy = y2 - y1;

        /* culadr */

        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*y1+x1/16;                   ❿
        dad = x1 % 16;

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );                     ⓫
        send_parameter( 16*dad );

        /* culpar */
        dc_reg = 3;                                     ⓬
        d_reg = dy;                                     ⓭
        d2_reg = dx;                                    ⓮
        d1_reg = -1;                                    ⓯
        dm_reg = dy;                                    ⓰

        /* vectw */
        send_command( 0x4c );                           ⓱
        send_parameter( 0x40 );                         ⓲
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );
        send_word_parameter( d2_reg );                  ⓳
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );                           ⓴
}
```

解説

❶描画モードの指定に使う定数をマクロ定義します。

❷VRAM1プレーンに対して四辺形を描画する関数box0( )を定義します。
❸仮引数の宣言です。
❹box0( )内で使用する変数を宣言します。
❺TEXTWコマンドの送出です。TEXTWのコマンドは78Hです。実線で描画するので，パラメータFFFFHを送っています。
❻ドット修正モードを指定するためWRITEコマンドを送出します。コマンドは23H(SET)，22H(CLEAR)，21H(COMPLEMENT)でパラメータはありません。
❼描画方向が0だけで済むように座標の処理を行います。まず，$x_1 > x_2$の場合，x_1とx_2を入れ替えます。
❽$y_1 > y_2$の場合，y_1とy_2を入れ替えます。
❾X方向の変位をdx，Y方向の変位をdyにセットします。
❿次にGDCの描画開始アドレス，ドットアドレスを計算し，変数ead，dadに入れます。描画するプレーンに応じてead_offsetに4000H，8000H，C000Hを設定しています。
⓫CSRWコマンドをGDCに送出します。コマンドは49H，パラメータはEAD，dADなどです。
⓬次にGDCの描画用パラメータレジスタの値を計算します。まず，DCレジスタに3をセットします。
⓭Dレジスタには第1描画方向変位をセットします。描画方向DIRは0としているので，dyが第1描画方向変位になります。
⓮D2レジスタには第2描画方向変位をセットします。D2レジスタにはdxを設定します。
⓯D1レジスタに−1をセットします。
⓰DMレジスタには第1描画方向変位数を設定します。つまりdyです。
⓱VECTWコマンドにより描画パラメータをGDCに送出します。VECTWコマンドは4CHです。
⓲四辺形描画で描画方向が0なので，40Hをパラメータ送出します。
⓳描画パラメータレジスタの値をそれぞれ送出します。
⓴VECTEコマンドを送って四辺形描画を開始させます。

これでVRAM1プレーンに対して描画する関数が定義できました。あとはこのbox0()関数を用いて一般の四辺形を描画する関数box()を定義します。box()関数は，リスト19のようになります。

◉リスト 19

```
/*
        四辺形を描画する関数
*/
void box( x1 , y1 , x2 , y2 , color )
int x1;            /*      左上のX座標      */
int y1;            /*      左上のY座標      */
int x2;            /*      右下のX座標      */
int y2;            /*      右下のY座標      */
int color;         /*      色               */
{
        if( (color & 1) != 0 )
                box0(x1,y1,x2,y2,0,PSET);
        else
                box0(x1,y1,x2,y2,0,PRESET);
        if( (color & 2) != 0 )
                box0(x1,y1,x2,y2,1,PSET);
        else
                box0(x1,y1,x2,y2,1,PRESET);
        if( (color& 4) != 0 )
                box0(x1,y1,x2,y2,2,PSET);
        else
                box0(x1,y1,x2,y2,2,PRESET);
}
```

指定した色コードに応じて描画するプレーンを選択しています。box関数をテストするプログラムをリスト20に示します。四辺形で画面全体を埋めるデモプログラムです。

実際問題として水平，垂直な直線の描画はGDCよりもCPUで行ったほうが高速に行えます。プログラムの速度が問題になるならば，CPUでVRAMを直接アクセスし四辺形を描画するルーチンを作成してください。

●リスト 20

```
/*
        四辺形を描く関数のテスト
*/
void main()
{
        int i;
        int x1,y1;
        int x2,y2;
        int color;

        start();
        all();
        cls2();
        for(i=300;i>=3;i--){
                x1 = 320 - i;
                y1 = 200 - i*400L/600;
                x2 = 320 + i;
                y2 = 200 + i*400L/600;
                color = i % 7 + 1;
                box( x1 , y1 , x2 , y2 , color );
        }
}
```

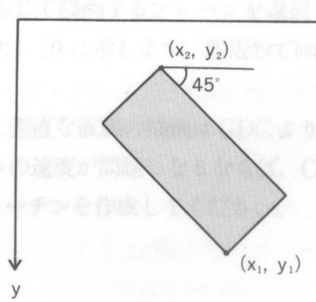
6 傾斜した四辺形描画

前節では，水平な四辺形を描画する関数を考えました。この節では，45°傾いた四辺形を描くことにします。水平な四辺形描画とは，VECTWの描画方向dirと第1描画方向ドット数，第2描画方向ドット数の計算が違ってきます。水平な四辺形に比べて，この計算は多少面倒です。図3-16のような傾斜四辺形を描くことにします。

2点(x_1, y_1), (x_2, y_2)を頂点にもつ四辺形は無数にありますが，辺の傾きが45°のものはひとつだけですから，ふたつの頂点座標を指定すれば四辺形を特定できます。そこで，次の関数を定義することをめざします。

```
/* 傾斜四辺形の描画 */
void box2( x1 , y1 , x2 , y2 , color )
int x1;          /* 描画を開始する点のX座標 */
int y1;          /* 描画を開始する点のY座標 */
int x2;          /* 対角のX座標 */
int y2;          /* 対角のY座標 */
int color;       /*     色     */
```

●図3-16●



なお，傾斜四辺形では，頂点座標から描画ドット数を計算するときに誤差が生じるため，描画開始点以外の頂点は真の位置と多少ずれますので，あまり幅の狭い四辺形を描くのには適しません。頂点の座標が厳密に描かれる必要がある場合は，直線描画を4回行ったほうがよいでしょう。

傾斜四辺形描画で送出するコマンドは，ほかの描画と同じです。

- TEXTW コマンド
- WRITE コマンド
- CSRW コマンド
- VECTW コマンド
- VECTE コマンド

1 TEXTW コマンド

傾斜四辺形のラインパターンです。実線を指定する場合の送出コマンド/パラメータは次のようになります。

コマンド	0AAH
パラメータ 1	0FFH
パラメータ 2	0FFH

2 WRITE コマンド

ドット修正モードの指定で四辺形描画などと同じです。送出コマンド/パラメータは，次のとおりです。

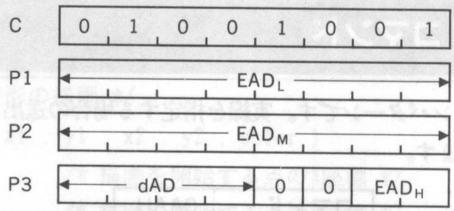
コマンド	20H	REPLACE モード時
	21H	COMPLEMENT モード時
	22H	CLEAR モード時
	23H	SET モード時
パラメータ	なし	

3 CSRW コマンド

傾斜四辺形を描画開始する点のアドレス/ドットアドレスを設定します。PC-9801の場合，描画を開始する点の座標を(x_1, y_1)とすれば，ほかの描画の場合と同様

```
ead = 0x4000 + 40 * y1 + x1/16;
dad = x1 % 16;
```

です。CSRWで送出するコマンド/パラメータも同様です。



4 VECTW コマンド

VECTWのパラメータは，四辺形描画時と違っています。
送出するコマンド/パラメータは，次ページ上のとおりです。 設定するパラメータの値は，次のとおりです。

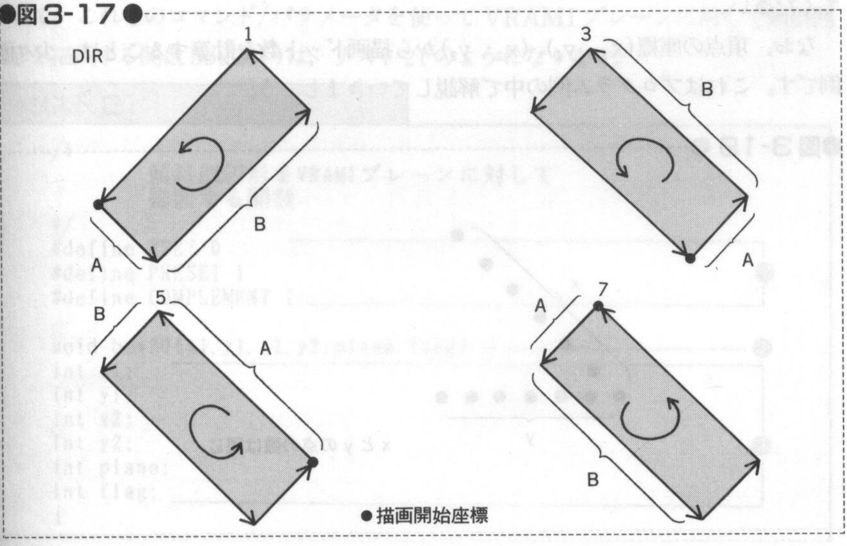
■ DIR

傾斜四辺形を描画する方向をDIRに設定します。

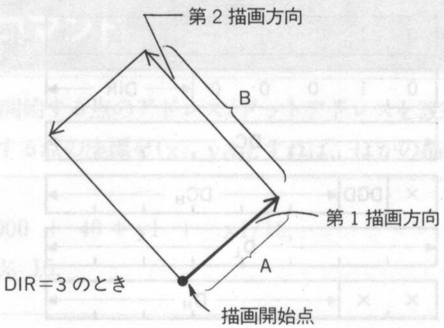
水平な四辺形を描画することと異なり，DIRには1，3，5，7の奇数をセットします。DIRの値と描画方向の対応は図3-17のとおりです。この図からわかるように，描画は左まわりに行われます。描画開始位置，描画ドット数をうまく設定することでひとつの描画方向で，すべての傾斜四辺形を描くことができます。

	2進数								16進数
C	0	1	0	0	1	1	0	0	4CH
P1	0	1	0	0	0	←DIR→			40H＋DIR
P2	←DC_L→								
P3	×	DGD	←DC_H→						
P4	←D_L→								
P5	×	×	←D_H→						
P6	←$D2_L$→								
P7	×	×	←$D2_H$→						
P8	←$D1_L$→								
P9	×	×	←$D1_H$→						
P10	←DM_L→								
P11	×	×	←DM_H→						

●図3-17●



●図 3-18●



■ DC レジスタ

3 を設定します。

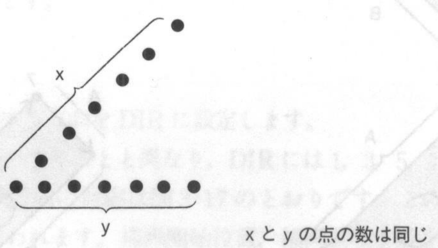
■ D レジスタ

第 1 描画方向の描画ドット数を指定します。

図 3-18 の中の A を指定します。傾斜した四辺形の場合，斜めの線のドット数を指定しますが，図 3-19 に示すように，これは水平な線と同じドット数なので注意してください。

なお，頂点の座標(x_1, y_1), (x_2, y_2)から描画ドット数を計算することは，少々面倒です。これはプログラム例の中で解説していきましょう。

●図 3-19●



■ D2 レジスタ

第2描画方向の変位数を指定します。図3-17でいえばBの大きさです。この値の計算もプログラム例の中で解説します。

■ D1 レジスタ

D1＝－1

を設定します。

■ DM レジスタ

第1描画方向のドット数を指定します。

5 VECTE コマンド

傾斜四辺形の描画を開始します。コマンド/パラメータは，次のとおりです。

コマンド	6CH
パラメータ	なし

さて，これらのコマンド/パラメータを使ってVRAM1プレーンに対して傾斜四辺形を描画する関数box20()は，リスト21のようになります。

●リスト21

```
/*
        傾斜四辺形をVRAM1プレーンに対して
        描画する関数
*/
#define PSET 0
#define PRESET 1                              ❶
#define COMPLEMENT 2

void box20(x1,y1,x2,y2,plane,flag) ─────────── ❷
int x1;
int y1;
int x2;
int y2;                                       ❸
int plane;
int flag;
{
```

```
unsigned ead,ead_offset;
unsigned dad;
int temp;
int dx,dy;
int a,b;

int dc_reg;        /* DC  レジスタ */
int d_reg;         /* D   レジスタ */
int d2_reg;        /* D2  レジスタ */
int d1_reg;        /* D1  レジスタ */
int dm_reg;        /* DM  レジスタ */


/* textw */
send_command( 0x78 );   /* textw command */
send_word_parameter( 0xffff );   /* pattern */

/* write */
if( flag == PSET )
        send_command( 0x23 );
if( flag == PRESET )
        send_command( 0x22 );
if( flag == COMPLEMENT )
        send_command( 0x21 );

if ( y1 < y2 ){
        temp = x1; x1 = x2; x2 = temp;
        temp = y1; y1 = y2; y2 = temp;
}

dx = x2 - x1;
dy = y2 - y1;

if( x1 < x2 ){  /* 長辺から描画 */
        a = ( abs( dx ) + abs( dy ) )/2;
        b = abs( abs( dx ) - abs( dy ) )/2;
}
if( x1 > x2){   /* 短辺から描画 */
        a = abs( abs( dx ) - abs( dy ) )/2;
        b = ( abs( dx ) + abs( dy ) )/2;
}
if( x1 == x2){
        a = ( abs( dx ) + abs( dy ) )/2;
        b = a;
}
```

```
        /* culadr */

        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*y1+x1/16;                        ⓭
        dad = x1 % 16;

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );                          ⓮
        send_parameter( 16*dad );

        /* culpar */
        dc_reg = 3;
        d_reg = a;
        d2_reg = b;                                          ⓯
        d1_reg = -1;
        dm_reg = a;

        /* vectw */
        send_command( 0x4c );
        send_parameter( 0x43 );        /* dir = 3 */
        send_word_parameter( dc_reg );
        send_word_parameter( d_reg );                        ⓰
        send_word_parameter( d2_reg );
        send_word_parameter( d1_reg );
        send_word_parameter( dm_reg );

        /* vecte */
        send_command( 0x6c );                                ⓱
}
```

傾斜四辺形は4つの描画方向が選択できます(図3-17)。しかし，うまく工夫すると，ひとつの描画方向ですべての傾斜四辺形が描けます。ここでは描画方向を3にして描画することを考えてみます。

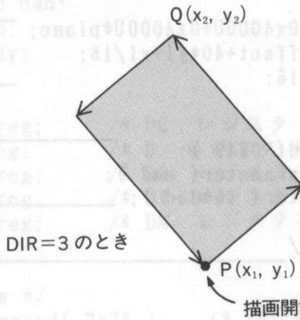
描画方向が3の場合，図3-20のような描画になります。

つまり，必ず一番下の頂点から描画が開始されます。傾斜四辺形の頂点をP(x_1, y_1)，Q(x_2, y_2)とします。このとき$y_1 > y_2$であることにします。そうでない場合はP，Qの座標を入れ替えます。そして，点Pから描画を開始すればよいわけです。

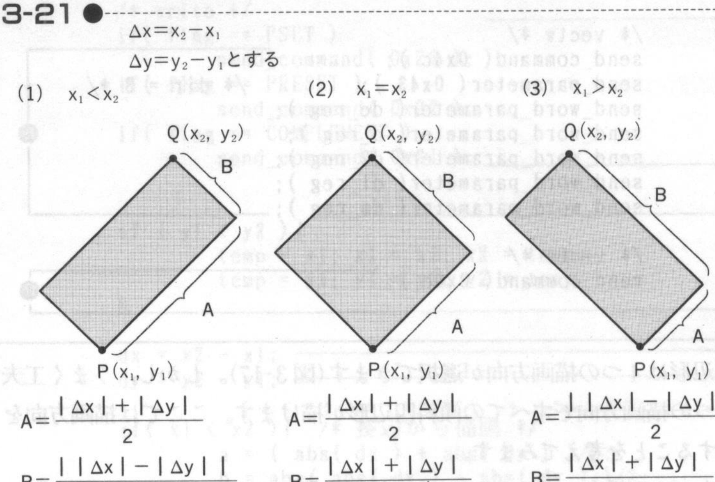
A，Bの値はP，Qの位置関係により，図3-21の3とおりになります。

このようにA，Bなどを設定すれば，描画方向を3に固定してもすべての四辺形を描画できます。わかりにくいかもしれないので，フローチャート(図3-22)を示しておきましょう。

●図 3-20●



●図 3-21●



それでは，リスト 21を解説していきます。

解説

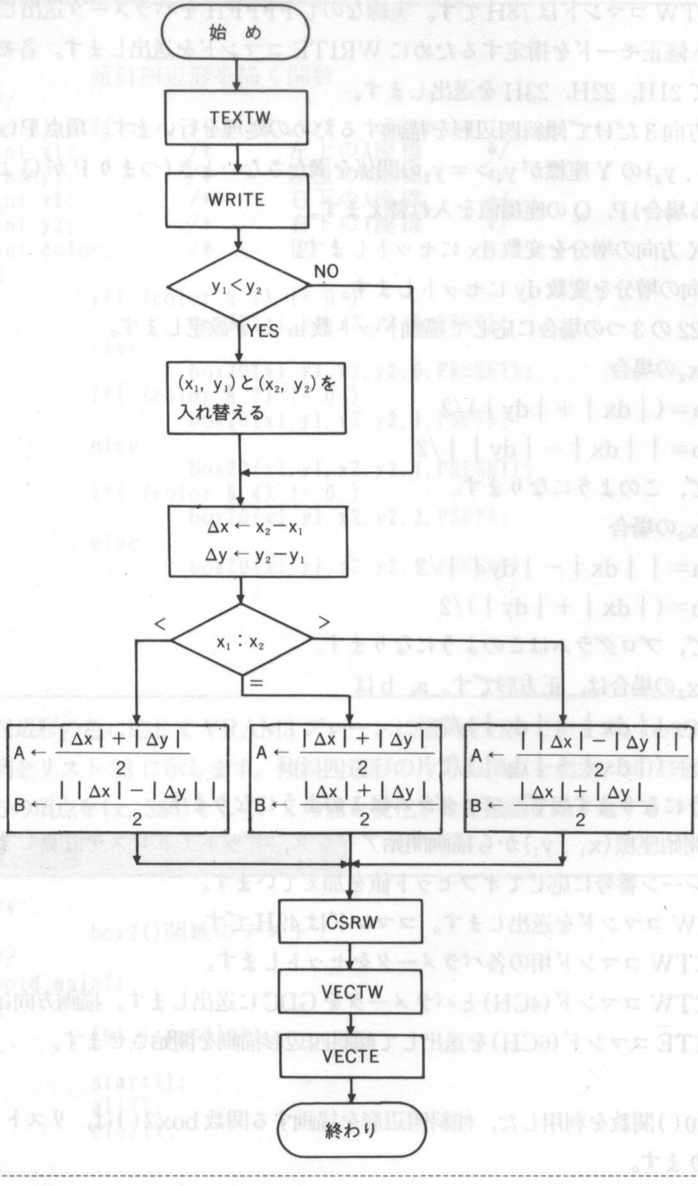
❶定数をマクロ定義します。

❷VRAM1プレーンに対して，傾斜四辺形を描画する関数box20()を定義します。

❸仮引数の定義です。

❹box20()関数内で使用するローカル変数の宣言をします。

●図 3-22●



始め
TEXTW
WRITE
y₁<y₂
NO
YES
(x₁, y₁)と(x₂, y₂)を入れ替える
Δx ← x₂−x₁
Δy ← y₂−y₁
<
>
x₁ : x₂
=
A ← (|Δx|+|Δy|)/2
B ← ||Δx|−|Δy||/2
A ← (|Δx|+|Δy|)/2
B ← (|Δx|+|Δy|)/2
A ← ||Δx|−|Δy||/2
B ← (|Δx|+|Δy|)/2
CSRW
VECTW
VECTE
終わり

❺ TEXTW コマンドを送出して傾斜四辺形描画時のラインパターンを指示します。TEXTW コマンドは 78H です。実線なので FFFFH をパラメータ送出します。

❻ドット修正モードを指定するために WRITE コマンドを送出します。各モードに応じて 21H，22H，23H を送出します。

❼描画方向 3 だけで傾斜四辺形を描画するための処理を行います。頂点 P(x_1 , y_1)，Q(x_2 , y_2)の Y 座標が $y_1 >= y_2$ の関係を満たさないとき(つまり P が Q よりも上にある場合) P，Q の座標値を入れ替えます。

❽次に X 方向の増分を変数 dx にセットします。

❾ Y 方向の増分を変数 dy にセットします。

図 3-22 の 3 つの場合に応じて描画ドット数 a，b を設定します。

❿ $x_1 < x_2$ の場合

$$a = (|dx| + |dy|)/2$$

$$b = ||dx| - |dy||/2$$

なので，このようになります。

⓫ $x_1 > x_2$ の場合

$$a = ||dx| - |dy||/2$$

$$b = (|dx| + |dy|)/2$$

なので，プログラムはこのようになります。

⓬ $x_1 = x_2$ の場合は，正方形です。a，b は

$$a = (|dx| + |dy|)/2$$

$$b = (|dx| + |dy|)/2$$

のようになりますので，プログラムはこのようになります。

⓭描画開始座標(x_1 , y_1)から描画開始アドレス，ドットアドレスを計算します。描画プレーン番号に応じてオフセット値を加えています。

⓮ CSRW コマンドを送出します。コマンドは 49H です。

⓯ VECTW コマンド用の各パラメータをセットします。

⓰ VECTW コマンド(4CH)とパラメータを GDC に送出します。描画方向は 3 です。

⓱ VECTE コマンド(6CH)を送出して傾斜四辺形描画を開始させます。

box20()関数を利用した，傾斜四辺形を描画する関数 box2()は，リスト 22 のようになります。

◉リスト22

```
/*
        傾斜四辺形を描く関数
*/
void box2( x1 , y1 , x2 , y2 , color )
int x1;         /*    左上のX座標       */
int y1;         /*    左上のY座標       */
int x2;         /*    右下のX座標       */
int y2;         /*    右下のY座標       */
int color;      /*    色                */
{
        if( (color & 1) != 0 )
                box20(x1,y1,x2,y2,0,PSET);
        else
                box20(x1,y1,x2,y2,0,PRESET);
        if( (color & 2) != 0 )
                box20(x1,y1,x2,y2,1,PSET);
        else
                box20(x1,y1,x2,y2,1,PRESET);
        if( (color & 4) != 0 )
                box20(x1,y1,x2,y2,2,PSET);
        else
                box20(x1,y1,x2,y2,2,PRESET);
}
```

傾斜四辺形の色に応じてVRAM3プレーンに描画を行っています。box2()関数の使用例をリスト23に示します。傾斜四辺形の片方の頂点を(320, 50)に固定し，もう一方の頂点を(x, 350)として，xの値を変化させて描画を繰り返すものです。

◉リスト23

```
/*
        box2()関数のテスト
*/
void main()
{
        int i,x,color;

        start();
        all();
        cls2();
```

```
        for(i=0;i<=88;i++){
            x = i*5+100;
            color = i % 7 +1;
            box2( 320 , 50 , x , 350 , color);
        }
}
```

7 グラフィックス文字描画

GDCは文字を描画する機能をもっています。これからのパーソナルコンピュータでは，文字をグラフィックス画面に描画する速さが重要になってくるでしょう。たとえば，MS-WINDOWSなどのウィンドウシステム上では，描画速度がウィンドウシステム全体の使い勝手を決めてしまいます。GDCのグラフィックス文字描画はあまり高速ではありませんが，いろいろおもしろい機能をもっています。速度だけが重要ならば，CPUで描画するプログラムを書くべきです。

ここでは，指定した座標に文字を描画する関数gputcを定義します。この関数で描画できるのは8×8ドットのASCII文字だけです。8×8ドット以外の文字(たとえば，漢字)を描画する関数は，このあとで解説します。

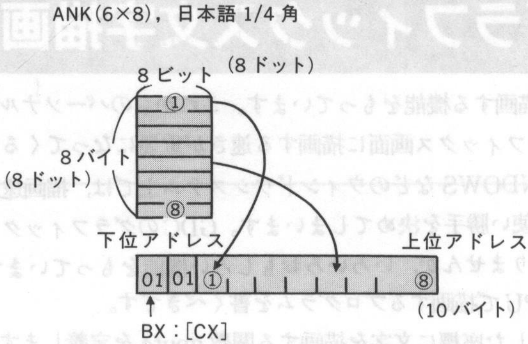
```
/* グラフィックス文字を描画する関数 */
gputc( ascii_code , x , y , color)
int ascii_code /* 描画する文字のアスキーコード */
int x;          /* 描画開始X座標 */
int y;          /* 描画開始Y座標 */
int color;      /* 色 */
```

グラフィックス文字を描画する場合，必要になるのが文字のフォントデータです。まず，フォントを読み出す関数set_pattern()を定義します。

```
void set_pattern( code , pattern )
int code;      /* 文字コード */
char pattern[]; /* フォントパターンを格納する配列 */
```

PC-9801はROM内にフォントをもっているのでこれを参照します。ROMから直接読み出すのは大変なので，BIOSを利用します。ROM BIOSの「フォントパターンの読み出し」を使用すると，簡単にフォントパターンが得られます。使い方は，次のとおりです。

●図 3-23●



AH	14H
BX	フォントパターンバッファの先頭アドレス(セグメントアドレス)
CX	フォントパターンバッファの先頭アドレス(オフセットアドレス)
DX	文字コード
INT	18H

ANK コードの場合

DL	ASCII コード	
DH	00H	6×7 フォント
	80H	7×13 フォント

バッファに展開されるデータ形式は，図 3-23 のとおりです。

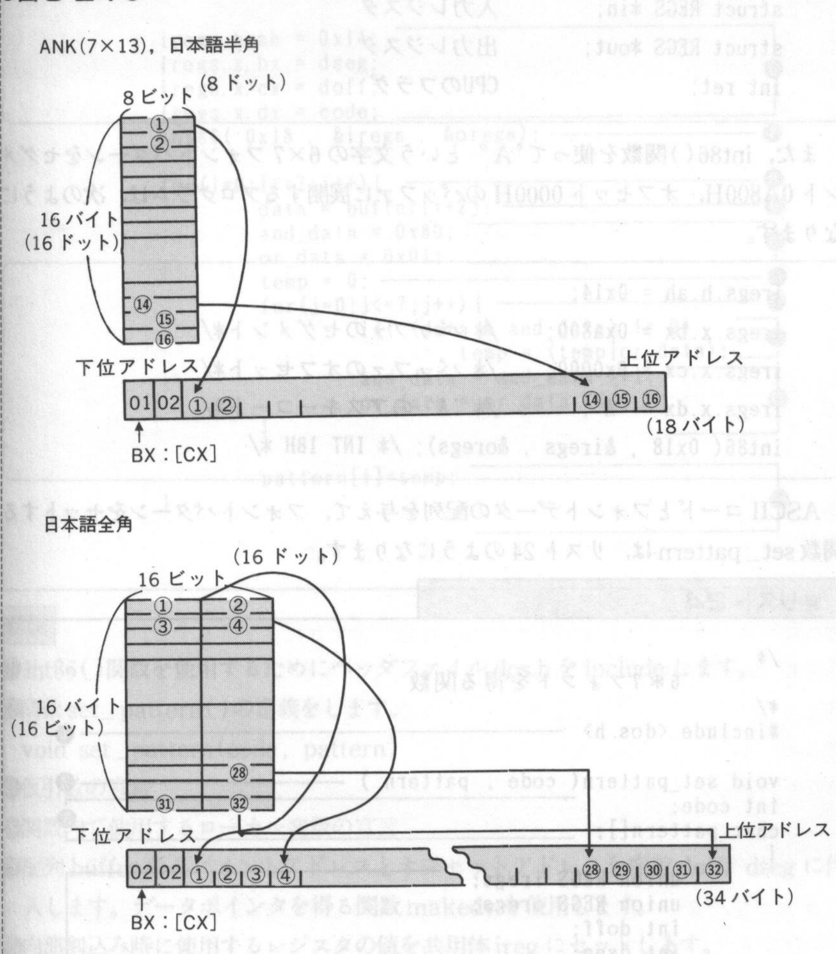
日本語コードの場合

DX	JIS コード

バッファに展開されるデータ形式は，図 3-24 のとおりです。

ここでは 8×8 の文字描画を考えているので，ANK コードの 6×7 フォントを利用します。BIOS は内部割り込みを使用していますが，C 言語から内部割り込みを起動するには，int86( )関数を使います。

●図 3-24 ●




```
#include <dos.h>
ret = int86(intnum,in,out);
int intnum;                    割込み番号
```

```
struct REGS *in;        入力レジスタ
struct REGS *out;       出力レジスタ
int ret;                CPUのフラグ
```

また，int86()関数を使って"A"という文字の6×7フォントパターンをセグメント0A800H，オフセット0000Hのバッファに展開するプログラムは，次のようになります。

```
iregs.h.ah = 0x14;
iregs.x.bx = 0xa800;    /* バッファのセグメント*/
iregs.x.cx = 0x0000;    /* バッファのオフセット*/
iregs.x.dx = 'A';       /* 'A' のアスキーコード*/
int86( 0x18 , &iregs , &oregs); /* INT 18H */
```

ASCIIコードとフォントデータの配列を与えて，フォントパターンをセットする関数set_patternは，リスト24のようになります。

●リスト24

```
/*
        6*7フォントを得る関数
*/
#include <dos.h>                                  ❶

void set_pattern( code , pattern )                ❷
int code;                                         ┐
char pattern[];                                   ┘❸
{
        union REGS iregs;                         ┐
        union REGS oregs;                         │
        int doff;                                 │
        int dseg;                                 │
        char buffer[34];                          │❹
        int i,j;                                  │
        unsigned char data;                       │
        unsigned char  and_data,or_data;          │
        unsigned char temp;                       ┘

        makedv(buffer,&doff,&dseg);               ❺
```

```
          iregs.h.ah = 0x14;       ─┐
          iregs.x.bx = dseg;        │❻
          iregs.x.cx = doff;        │
          iregs.x.dx = code;       ─┘
          int86( 0x18 , &iregs , &oregs);  ── ❼

          for(i=0;i<=7;i++){  ── ❽
                  data = buffer[i+2];  ── ❾
                  and_data = 0x80;     ─┐❿
                  or_data = 0x01;      ─┘
                  temp = 0;  ── ⓫
                  for(j=0;j<=7;j++){  ── ⓬
                          if( (data & and_data) != 0)   ─┐⓭
                                  temp = (temp|or_data); ─┘
                          and_data = and_data >>1;   ─┐⓮
                          or_data = or_data <<1;      │
                  }                                  ─┘

                  pattern[i]=temp;   ─┐⓯
          }                           │
}
```

解説

❶int86( )関数を使用するためにヘッダファイル dos.h を include します。

❷関数 set_ pattern( )の定義をします。

void set_ pattern(code, pattern)

❸仮引数の宣言

❹関数内で使用するローカル変数の宣言

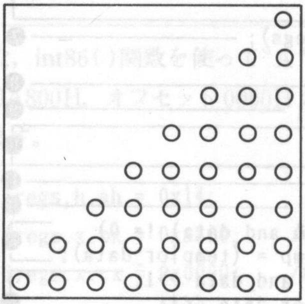
❺配列 buffer のセグメントアドレスとオフセットアドレスを変数 doff, dseg に代入します。データポインタを得る関数 makedv を使用します。

❻内部割込み時に使用するレジスタの値を共用体 ireg にセットします。

❼int86( )関数で内部割込み 18H を起動します。

❽これで配列 buffer にフォントデータが得られました。BIOS で得られるフォントデータは図 3-25 の①の形式ですが, GDC が必要とするのは図 3-25 の②の形式です。ビットパターンの LSB と MSB が逆になっているのがわかります。そこで, ビットパターンを逆にする処理が必要になります。以下がその部分です。
ビットパターンの反転を 8 回行います。

●図 3-25●



ROM BIOS;	GDCが必要とするデータ
01H	80H
03H	C0H
07H	E0H
0FH	F0H
1FH	F8H
3FH	FCH
7FH	FEH
FFH	FFH
①	②

❾変数 data にi番目のフォントパターンをセットします。buffer の先頭2バイトはフォントパターンではないので読み飛ばしています。

❿変数 and_data, or_data をそれぞれ80H, 01H に設定します。

⓫逆順のビットパターンを得る変数 temp を，まず0にクリアします。

⓬変数 data の内容をbit7からbit0まで調べて行きます。

⓭もし data の(7-j)ビット目が立っていたならば，変数 temp の j ビットを立てます。

⓮and_data, or_data をシフトします。

⓯これを j=0, 1, 2, …, 7まで行えば，変数 temp にビットパターンの逆順がセットされます。それを pattern [i] に代入します。

フォントパターンを得る関数が定義できたので，グラフィックス文字描画を行うことにします。グラフィックス文字描画を行うときに GDC に送出すべきコマンドは，次のとおりです。

- TEXTW コマンド
- WRITE コマンド
- CSRW コマンド
- VECTW コマンド
- TEXTE コマンド

1 TEXTW コマンド

TEXTW コマンドの意味はグラフィックス文字描画の場合，ほかの描画とは大きく違っています。TEXTWコマンドの役割はフォントパターンをGDC に知らせることです。

コマンド/パラメータは，次のようになります。

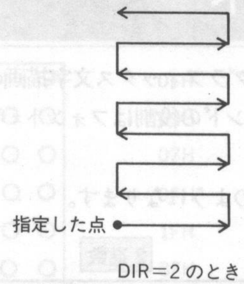


2 WRITE コマンド

ドット修正モードを指定します。コマンド/パラメータは，次のようになります。

コマンド	20H	REPLACE モード時
	21H	COMPLEMENT モード時
	22H	CLEAR モード時
	23H	SET モード時
パラメータ	なし	

●図 3-26●



3 CSRW コマンド

グラフィックス文字を描画開始する点のアドレス/ドットアドレスを設定します。描画開始点と実際に描画されるデータの関係を図 3-26 に示します。

VRAM の横幅が 40 ワードなので，(x, y)から EAD/dAD は

```
ead = 0x4000 + 40 * y + x/16;
dad = x % 16;
```

で計算されます。プレーン 0 に描画の場合に送出するコマンド/パラメータは，次のとおりです。

	2 進数								16 進数
C	0	1	0	0	1	0	0	1	49H
P1	←EAD_L→								
P2	←EAD_M→								
P3	←dAD→				0	0	EAD_H		

4 VECTW コマンド

VECTW は，描画の種類と描画方向の設定のために使用されます。送出するコマ

ンド/パラメータは，次のとおりです。

	2進数								16進数
C	0	1	0	0	1	1	0	0	4CH
P1	0	0	0	1	0	← DIR →			
P2	← DC_L →								
P3	×	DGD	← DC_H →						

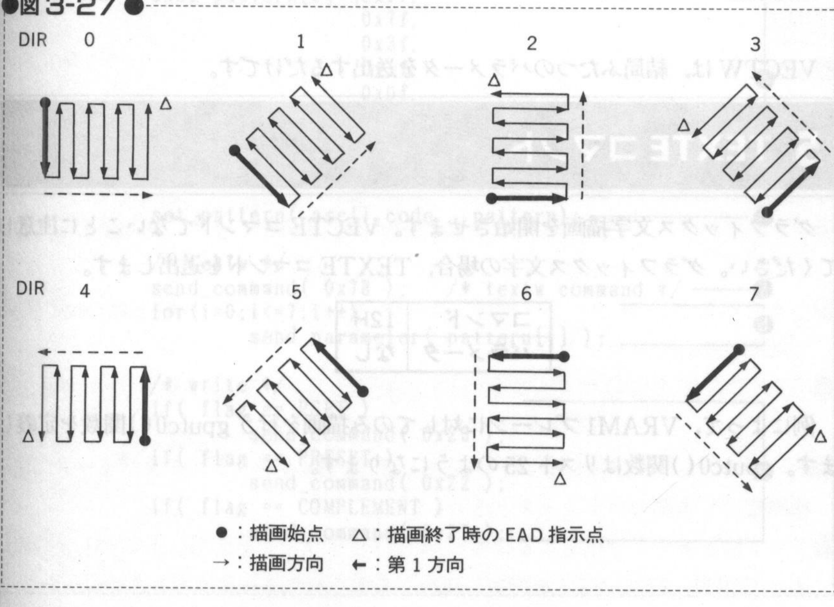
パラメータの値は，次のようになります。

■ DIR

グラフィックス文字の描画方向を指定します。

DIRに奇数を設定すれば，斜めに傾いた文字を描画することができます。ただし，大きさは傾かない場合の1.41倍になります(長さの比)。描画は，図3-27のように描画方向を180°変えながら行われます。

●図3-27●



GDCの描画パラメータは，次の値を設定します。

■ DCレジスタ

7を設定します。

■ Dレジスタ

設定禁止

■ D2レジスタ

設定禁止

■ D1レジスタ

設定禁止

■ DMレジスタ

設定禁止

VECTWは，結局ふたつのパラメータを送出するだけです。

5 TEXTEコマンド

グラフィックス文字描画を開始させます。VECTEコマンドでないことに注意してください。グラフィックス文字の場合，TEXTEコマンドを送出します。

コマンド	12H
パラメータ	なし

例によって，VRAM1プレーンに対してのみ描画を行うgputc0( )関数を定義します。gputc0( )関数はリスト25のようになります。

● リスト 25

```
/*
 グラフィックス文字描画
*/

#define PSET 0
#define PRESET 1                                    ❶
#define COMPLEMENT 2

void gputc0(ascii_code,x,y,plane,flag)              ❷
int ascii_code;
int x;
int y;                                              ❸
int plane;
int flag;
{
      unsigned ead,ead_offset;
      unsigned dad;
      int i;                                        ❹

      int dc_reg;       /* DC  レジスタ */

      /* フォントデータ格納配列 */
      char pattern[8]={0xff,
                       0x7f,
                       0x3f,
                       0x1f,
                       0x0f,                        ❺
                       0x07,
                       0x03,
                       0xff};

      set_pattern( ascii_code , pattern);           ❻

      /* textw */
      send_command( 0x78 );   /* textw command */   ❼
      for(i=0;i<=7;i++)
            send_parameter( pattern[i] );           ❽

      /* write */
      if( flag == PSET )
            send_command( 0x23 );
      if( flag == PRESET )
            send_command( 0x22 );                   ❾
      if( flag == COMPLEMENT )
            send_command( 0x21 );
```

```
        /* culadr */
        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*y+x/16;                    ⑩
        dad = x % 16;

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );                    ⑪
        send_parameter( 16*dad );
        dc_reg = 7;                                    ⑫

        /* TEXTE */
        send_command( 0x4c );                          ⑬
        send_parameter( 0x12 );       /* dir = 2 */    ⑭
        send_word_parameter( dc_reg );                 ⑮

        /* tecte */
        send_command( 0x68 );                          ⑯
}
```

解説

❶定数をマクロ定義します。

❷関数gputc0( )の宣言部です。

❸仮引数です。

❹ gputc0( )関数内で使用するローカル変数を宣言します。

❺フォントパターンを格納する配列 pattern [ ] を宣言します。初期値として3角形のパターンを入れています。

❻与えられた ASCII コードに対応するフォントパターンを，set_ pattern( )関数で配列 pattern にセットします。

❼ TEXTW で文字のパターンを GDC に転送します。TEXTW コマンドは 78H です。

❽次にパターンデータをパラメータ送出します。8×8 ドットのグラフィックス文字描画なので 8 個のパラメータを送出します。

❾ドット修正モードを指定するために WRITE コマンドを送出します。flag の値によって 21H，22H，23H を選択し送出します。パラメータはありません。

⑩ X, Y座標から描画開始アドレス, ドットアドレスを計算します。
⑪CSRWコマンドを発行します。CSRWコマンドは49Hです。パラメータにはいま計算したEAD, dADを送出します。
⑫グラフィックス文字描画の場合, 描画パラメータレジスタはDCだけを使います。DCレジスタには7を設定します。
⑬VECTWコマンドの送出です。コマンドは, 4CHです。
⑭描画方向に2を指定しています。したがって, 第1パラメータは12Hです。
⑮第2パラメータはDCレジスタの値を送出します。
⑯最後に, TEXTEコマンド(68H)を送出してグラフィックス文字描画を開始させます。

VRAM1プレーンに対して文字を描画する関数gputc0()が定義できました。これを使うと, gputc()関数は, 次のようになります(リスト26)。

◉リスト26

```
/*
        グラフィックス文字描画
*/
gputc(ascii_code,x,y,color)
int ascii_code; /* アスキーコード */
int x;          /* 文字の左下のX座標   */
int y;          /* 文字の左下のY座標   */
int color;      /*     色            */
{
        if( (color & 1) != 0 )
                gputc0(ascii_code,x,y,0,PSET);
        else
                gputc0(ascii_code,x,y,0,PRESET);
        if( (color & 2) != 0 )
                gputc0(ascii_code,x,y,1,PSET);
        else
                gputc0(ascii_code,x,y,1,PRESET);
        if( (color & 4) != 0 )
                gputc0(ascii_code,x,y,2,PSET);
        else
                gputc0(ascii_code,x,y,2,PRESET);
}
```

描画色に応じて，gputc0()関数を3回呼び出しています。gputc()関数を使ったプログラム例をリスト27に示します。

◉リスト27

```
/*
        gputc()関数のテスト
*/
void main()
{
        int i,x,color;

        start();
        all();
        cls2();

        gputc( 'P' , 320 , 200 , 1);
        gputc( 'e' , 328 , 200 , 2);
        gputc( 'r' , 336 , 200 , 3);
        gputc( 's' , 344 , 200 , 4);
        gputc( 'o' , 352 , 200 , 5);
        gputc( 'n' , 360 , 200 , 6);
        gputc( 'a' , 368 , 200 , 7);
        gputc( 'l' , 377 , 200 , 1);

        gputc( 'C' , 320 , 208 , 1);
        gputc( 'o' , 328 , 208 , 2);
        gputc( 'm' , 336 , 208 , 3);
        gputc( 'p' , 344 , 208 , 4);
        gputc( 'u' , 352 , 208 , 5);
        gputc( 't' , 360 , 208 , 6);
        gputc( 'e' , 368 , 208 , 7);
        gputc( 'r' , 377 , 208 , 1);
}
```

〝Personal Computer〞という文字を色を変えながら描画するプログラムです。

8 ズーム機能と傾斜グラフィックス文字描画

GDCはズーム機能をもっています。GDCのズームには2種類の機能があります。ひとつはズームリード機能，もうひとつはズームライト機能です。ズームリード機能とは，VRAMのデータを拡大して表示する機能です。VRAMのデータは変化しません。ズームライト機能は描画するときに拡大するもので，VRAMに書かれるデータ自体が大きくなります。ズームリード機能は，ドットクロックを遅延させる外付けハードウェアが必要なため，PC-9801では残念ながら使えません。しかし，ズームライトは行えます。この節では，ズームライト機能を使ってグラフィックス文字を拡大することを考えてみます。また，あわせて文字描画方向を変えたり，傾斜グラフィックス文字描画も行うことにします。そこで，次の関数を定義します。

```
/* グラフィックス文字描画関数(拡大, 傾斜可能) */
gputc2( pattern , x , y , color , zoom , dir , slant)
char pattern[]; /* 描画する文字のフォントパターン */
int x;          /* 描画開始X座標 */
int y;          /* 描画開始Y座標 */
int color;      /* 色 */
int zoom;       /* 拡大率 */
int dir;        /* 描画方向 */
int slant;      /* 傾斜させるかどうか */
```

1 ZOOMコマンドについて

GDCにズームを行わせる場合には，ZOOMコマンドを送出します。一度，ZOOMコマンドを発行すると，以後ZOOMコマンドが送出されるまで，その拡大率が維持されます(描画のたびに送出する必要はありません)。ZOOMのコマン

ド，パラメータは，次のとおりです。

	2進数								16進数
C	0	1	0	0	0	1	1	0	46H
P	ZR				ZW				

拡大描画の拡大率は，ZWにセットします。

ZW	拡大率
0	1
1	2
2	3
⋮	⋮
15	16

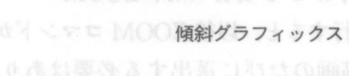
このZOOMコマンドで，1～16倍までの任意の拡大が行えるわけです。なお，PC-9801の場合，ズームリードはできないので，ZRは0に設定します。

2 傾斜グラフィックス文字描画について

GDCは傾斜したグラフィックス文字を描くことができます。傾斜グラフィックス文字とは図3-28のような描画をいいます。

文字が上にいくほど右にずれた形になります。描画方向を斜めにしたのとは，また違った描画です。傾斜グラフィックス文字はなかなかおもしろい機能ですが，画面の縦横比が1対1の場合は，ちょっと傾斜しすぎた感じになります。縦横比が2対1ぐらいだとちょうどよい感じです。傾斜グラフィックス文字は640×200モードで使用するのがよいでしょう。

●図3-28●



ズーム機能，傾斜グラフィックス文字描画機能をもった関数gputc2()はリスト28のようになります。

◉リスト 28

```
/*
 グラフィックス文字描画
  拡大，傾斜可能
*/

#define PSET 0
#define PRESET 1
#define COMPLEMENT 2
#define TRUE -1
#define FALSE 0

void gputc20( pattern , x , y , plane , flag , zoom , dir , slant)
char pattern[]; /* 描画する文字のフォントパターン */
int x;          /* 描画開始X座標 */
int y;          /* 描画開始Y座標 */
int plane;      /* 描画プレーン番号 */
int flag;       /* ドット修正モード */
int zoom;       /* 拡大率 */
int dir;        /* 描画方向 */
int slant;      /* 傾斜させるかどうか */
{
        unsigned ead,ead_offset;
        unsigned dad;
        int i;

        int dc_reg;     /* DC  レジスタ */

        /* zoom */
        send_command( 0x46 );       ─┐
        send_parameter( zoom );     ─┘❶

        /* textw */
        send_command( 0x78 );   /* textw command */
        for(i=0;i<=7;i++)
                send_parameter( pattern[i] );

        /* write */
        if( flag == PSET )
                send_command( 0x23 );
        if( flag == PRESET )
                send_command( 0x22 );
```

```
        if( flag == COMPLEMENT )
                send_command( 0x21 );

        /* culadr */

        ead_offset=0x4000U+0x4000U*plane;
        ead = ead_offset+40*y+x/16;
        dad = x % 16;

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( 16*dad );

        dc_reg = 7;
        /* vectw */
        send_command( 0x4c );                    ──❷

        if( slant == TRUE )
                send_parameter( 0x90+dir );
        else                                        ❸
                send_parameter( 0x10+dir );

        send_word_parameter( dc_reg );

        /* texte */
        send_command( 0x68 );
}
void gputc2(pattern,x,y,color,zoom,dir,slant)
char pattern[]; /* 描画する文字のフォントパターン */
int x;          /* 描画開始X座標 */
int y;          /* 描画開始Y座標 */
int color;      /* 色 */
int zoom;       /* 拡大率 */
int dir;        /* 描画方向 */
int slant;      /* 傾斜させるかどうか */

{
        if( (color & 1) != 0 )
           gputc20(pattern,x,y,0,PSET,zoom,dir,slant);
        else
           gputc20(pattern,x,y,0,PRESET,zoom,dir,slant);
        if( (color & 2) != 0 )
           gputc20(pattern,x,y,1,PSET,zoom,dir,slant);
```

```
    else
        gputc20(pattern, x, y, 1, PRESET, zoom, dir, slant);
    if( (color & 4) != 0 )
        gputc20(pattern, x, y, 2, PSET, zoom, dir, slant);
    else
        gputc20(pattern, x, y, 2, PRESET, zoom, dir, slant);
}
```

解説

❶gputc()関数との違いはZOOMコマンドの送出とVECTWのパラメータです。TEXTWコマンドを送出する前にZOOMコマンド46Hと拡大率を送出しています。

❷VECTWの部分はまずコマンド4CHを送出します。

❸変数slantに応じて90H＋dirか10H＋dirをパラメータ送出しています。

❹ほかの部分は，ほぼgputc()と同じです。

gputc2()関数では1文字ずつ座標や色を指定しなければならないので，文字列を表示するには不向きです。そこで，文字列を表示するための関数zprintf()を定義します。

たとえば，〝PC-9801〞という文字列を(320, 200)の位置に赤色で4倍の拡大率で傾斜せずに描画する場合

zprintf("PC-9801", 320, 200, 2, 4－1, FALSE);

のようになります。zprintf()関数の定義は，リスト29のとおりです。

◉リスト29

```
void zprintf(string, x, y, color, zoom, slant)
char string[];
int x;
int y;
int color;
int zoom;
int slant;
{
        int i;
```

```
    char pattern[8];
    int xpos;
    char c;

    for(i=0;i <= strlen( string )-1;i++){
        c = string[i];
        set_pattern(c,pattern);
        xpos = x + i*8*(zoom+1);
        gputc2(pattern,xpos,y,color,zoom,2,slant);
    }
}
```

zprintf()関数を使ったテストプログラムを，リスト30に示します。リスト30は文字の大きさを変えながら傾斜しないグラフィックス文字と傾斜したグラフィックス文字を描くものです。

◉リスト30

```
void main()
{
    start();
    all();
    cls2();

    zprintf("ZOOM=0 ",      100, 50,7,0,FALSE);
    zprintf("ZOOM=0 SLANT", 100,100,7,0,TRUE);

    zprintf("ZOOM=1 ",      100,150,2,1,FALSE);
    zprintf("ZOOM=1 SLANT", 100,200,2,1,TRUE);

    zprintf("ZOOM=2 ",      100,250,3,2,FALSE);
    zprintf("ZOOM=2 SLANT", 100,300,3,2,TRUE);

    zprintf("ZOOM=3 ",      100,350,4,3,FALSE);
    zprintf("ZOOM=3 SLANT", 100,399,4,3,TRUE);
}
```

9 グラフィック文字描画(漢字)

グラフィック文字描画機能を使って漢字を表示することもできます。漢字は通常16×16ドットで表します。ところが，GDCは8×8ドットの文字しか描画できません。そこで8×8ドットの文字描画を4回繰り返して漢字を描くことにします。漢字のフォントは，第7節で解説したBIOSの機能で読み出すことができます。ただし，漢字のフォントは16×16ドットなのでデータの量も32バイトと多くなります。また，BIOSで指定できるのはJISコードです。MS-DOSで一般に使われているのはシフトJISコードなので，シフトJISからJISコードに変換するプログラムが必要です。一般のテキストファイルは漢字だけでなく半角，1/4角の文字も含んでいます。それらもすべて表示できるようにします。この節は，次のような関数を定義することを目的とします。

```
/* 日本語の文字列を描画する関数(拡大・傾斜可能) */
void jprintf(string,x,y,color,zoom,slant);
char *string;   /* 文字列へのポインタ */
int x;          /* 描画開始X座標 */
int y;          /* 描画開始Y座標 */
int color;      /* 色 */
int zoom;       /* 拡大率 */
int slant;      /* 傾斜させるかどうか */
```

1/4角，半角，全角の3種類の文字を扱うので3種類のフォントを読み出す関数を定義する必要があります。この関数を次のように定義します。

```
x=set_pattern2( jiscode , pattern );
int code;        /* JISコード(ASCIIコードを含む) */
char pattern[]; /* フォントを展開する配列 */
int x;           /* 次の文字のX座標 */
```

set_ pattern2( )のプログラムは，リスト31のようになります。

●リスト31

```
/*
      フォントデータの読み出し
      漢字対応
*/
#include <dos.h>                                          ❶

int set_pattern2( code , pattern )                        ❷
int code;
char pattern[];                                           ❸
{
      union REGS iregs;
      union REGS oregs;
      int doff;
      int dseg;
      char buffer[34];                                    ❹
      int i,j;
      unsigned char data;
      unsigned char  and_data,or_data;
      unsigned char temp;

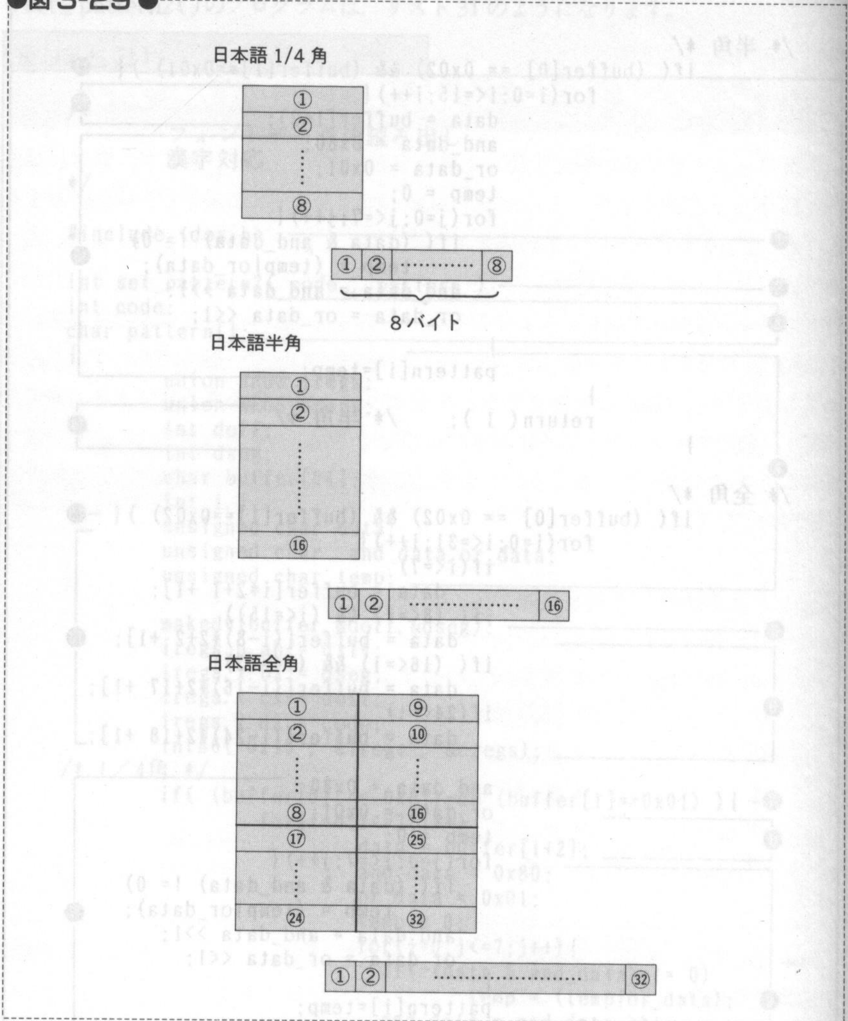
      makedv(buffer,&doff,&dseg);                         ❺
      iregs.h.ah = 0x14;
      iregs.x.bx = dseg;
      iregs.x.cx = doff;                                  ❻
      iregs.x.dx = code;
      int86( 0x18 , &iregs , &oregs);
/* 1／4角 */
      if( (buffer[0] == 0x01) && (buffer[1]==0x01) ){     ❼
            for(i=0;i<=7;i++){                            ❽
                  data = buffer[i+2];
                  and_data = 0x80;
                  or_data = 0x01;
                  temp = 0;
                  for(j=0;j<=7;j++){
                        if( (data & and_data) != 0)
                              temp = (temp|or_data);      ❾
                        and_data = and_data >>1;
                        or_data = or_data <<1;
                  }
                  pattern[i]=temp;
            }
            return( 0 );      /* 1／4角 */                ❿
      }
```

```
/* 半角 */
    if( (buffer[0] == 0x02) && (buffer[1]==0x01) ){ —⑪
            for(i=0;i<=15;i++){                        ⑫
                    data = buffer[i+2];
                    and_data = 0x80;
                    or_data = 0x01;
                    temp = 0;
                    for(j=0;j<=7;j++){
                       if( (data & and_data) != 0)
                           temp = (temp|or_data);      ⑬
                       and_data = and_data >>1;
                       or_data = or_data <<1;
                    }
                    pattern[i]=temp;
            }
            return( 1 );    /* 半角 */                 ⑭
    }

/* 全角 */
    if( (buffer[0] == 0x02) && (buffer[1]==0x02) ){ —⑮
            for(i=0;i<=31;i++){
                    if(i<=7)
                         data = buffer[i*2+1 +1];
                    if( (8<=i) && (i<=15))
                       data = buffer[(i-8)*2+2 +1];    ⑯
                    if( (16<=i) && (i<=23))
                       data = buffer[(i-16)*2+17 +1];
                    if(24<=i)
                       data = buffer[(i-24)*2+18 +1];

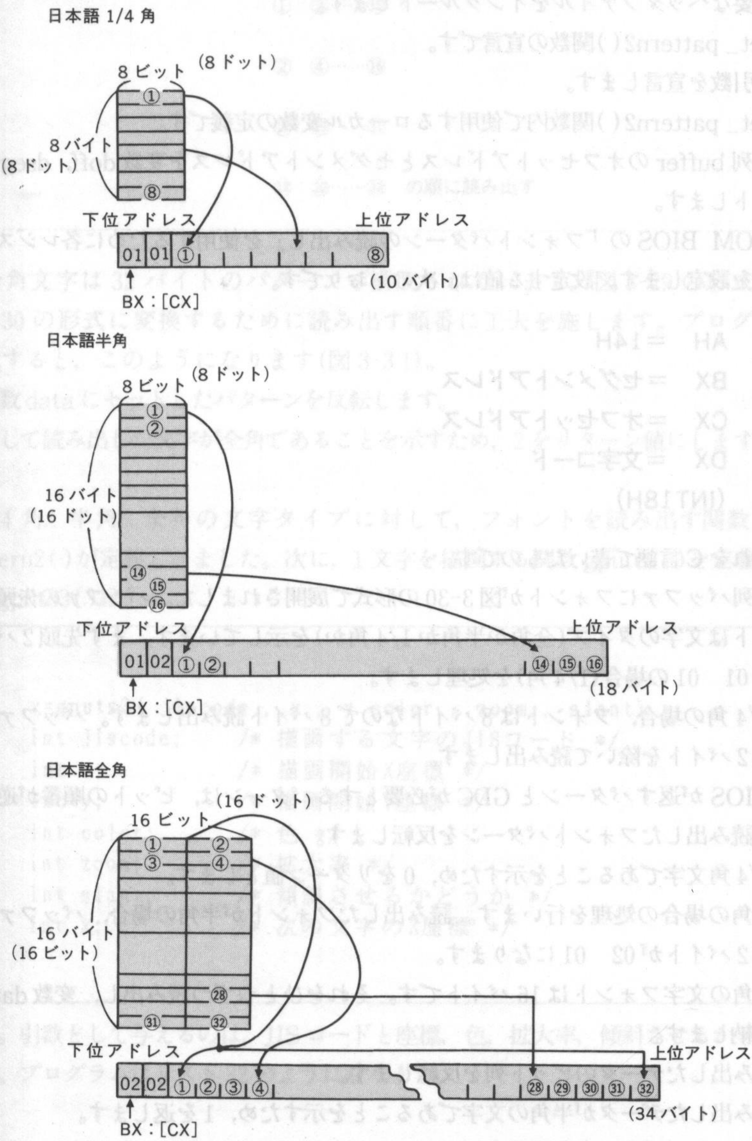
                    and_data = 0x80;
                    or_data = 0x01;
                    temp = 0;
                    for(j=0;j<=7;j++){
                       if( (data & and_data) != 0)
                            temp = (temp|or_data);     ⑰
                       and_data = and_data >>1;
                       or_data = or_data <<1;
                    }
                    pattern[i]=temp;
            }
            return( 2 );    /* 全角 */                 ⑱
    }
}
```

●図 3-29●



set_pattern2( )関数が展開するフォントパターンは，図 3-29 のようになります。BIOS の展開するフォントパターンは図 3-30 なので，これを変換する操作が必要になります。

●図 3-30●


日本語 1/4 角
8 ビット
(8 ドット)
8 バイト
(8 ドット)
下位アドレス
上位アドレス
01 01
(10 バイト)
BX : [CX]
日本語半角
8 ビット
(8 ドット)
16 バイト
(16 ドット)
下位アドレス
上位アドレス
01 02
(18 バイト)
BX : [CX]
日本語全角
(16 ドット)
16 ビット
16 バイト
(16 ビット)
下位アドレス
上位アドレス
02 02
(34 バイト)
BX : [CX]

解説

❶必要なヘッダファイルをインクルードします。

❷set_pattern2()関数の宣言です。

❸仮引数を宣言します。

❹set_pattern2()関数内で使用するローカル変数の定義です。

❺配列 buffer のオフセットアドレスとセグメントアドレスを変数 doff, dseg にセットします。

❻ROM BIOS の「フォントパターンの読み出し」を使用するために各レジスタの値を設定します。設定する値は，次のとおりです。

```
AH  =14H
BX  =セグメントアドレス
CX  =オフセットアドレス
DX  =文字コード
(INT18H)
```

これをC言語で書いたものです。

❼配列バッファにフォントが図 3-30 の形式で展開されました。バッファの先頭2バイトは文字のタイプ(全角か半角か 1/4 角か)を示しています。まず先頭2バイトが 01　01 の場合(1/4 角)を処理します。

❽1/4 角の場合，フォントは8バイトなので8バイト読み出します。バッファの先頭2バイトを除いて読み出します。

❾BIOS が返すパターンと GDC が必要とするパターンは，ビットの順番が逆なので読み出したフォントパターンを反転します。

❿1/4 角文字であることを示すため，0をリターン値とします。

⓫半角の場合の処理を行います。読み出したフォントが半角の場合，バッファの先頭2バイトが 02　01 になります。

⓬半角の文字フォントは 16 バイトです。それをひとつずつ読み出し，変数 data に格納します。

⓭読み出したデータのビット列を反転します。

⓮読み出したデータが半角の文字であることを示すため，1を返します。

⓯全角の処理です。全角の場合，バッファの先頭2バイトは 02　02 です。

●図 3-31 ●

① ③……⑮

② ④……⑯

⑰ ⑲……㉛

⑱ ⑳……㉜ の順に読み出す

⑯全角文字は 32 バイトのパターンで構成されています。図 3-29 の形式を図 3-30 の形式に変換するために読み出す順番に工夫を施します。プログラム化すると，このようになります(図 3-31)。

⑰変数 data にセットしたパターンを反転します。

⑱そして読み出した文字が全角であることを示すため，2 をリターン値にします。

1/4 角，半角，全角の文字タイプに対して，フォントを読み出す関数 set_pattern2( )が定義できました。次に，1 文字を描画する関数 gputc3( )を定義します。gputc3( )は

```
x=gputs3( jiscode , x, y color , zoom , slant)
int jiscode;        /* 描画する文字のJISコード */
int x;              /* 描画開始X座標 */
int y;              /* 描画開始Y座標 */
int color;          /* 色 */
int zoom;           /* 拡大率 */
int slant;          /* 傾斜させるかどうか */
int x;              /* 次の文字のX座標 */
```

です。引数として与えるのは，JIS コードと座標，色，拡大率，傾斜させるかどうかです。プログラムはリスト 32 のようになります。

◉リスト 32

```
/*
        JISコードに対応する1文字を
        描画する関数
*/
int gputc3( jiscode , x , y , color, zoom ,slant) ――― ❶
int jiscode;     /* 描画する文字のJISコード */
int x;           /* 描画開始X座標 */
int y;           /* 描画開始Y座標 */                    ❷
int color;       /* 色 */
int zoom;        /* 拡大率 */
int slant;       /* 傾斜させるかどうか */
{
        int type;
        char pattern[32];                              ❸
        int ofst;
        int sofst;

        type=set_pattern2( jiscode , pattern ); ――― ❹
        ofst= 8*(zoom+1); ――― ❺
        sofst = 8*(-slant)*(zoom+1); ――― ❻

        if( type == 0){ ――― ❼
                gputc2( pattern ,x , y , color , zoom ,  ❽
                        2 , slant );
                return( x+ofst );                       ❾
        }
        if( type == 1){ ――― ❿
                gputc2( pattern ,x+sofst , y-ofst ,     ⓫
                        color , zoom , 2 , slant );
                gputc2( pattern+8, x , y, color , zoom , ⓬
                        2 , slant );
                return(x+ofst);                         ⓭
        }
        if( type == 2){ ――― ⓮
                gputc2( pattern ,x+sofst , y-ofst ,
                        color , zoom , 2 , slant );
                gputc2( pattern+8,x+ofst+sofst, y-ofst ,
                        color , zoom , 2 , slant );
                gputc2( pattern+16 ,x , y , color ,      ⓯
                        zoom , 2 , slant );
                gputc2( pattern+24 ,x+ofst, y , color ,
                        zoom, 2, slant );
                return(x+ofst*2);                       ⓰
        }
}
```

解説

❶関数 gputc3( )を定義します。gputc3 は次の文字の X 座標を返すようにしたいので，int 型で宣言します。

❷仮引数の宣言です。

❸ gputc3( )関数の定義の中で使用するローカル変数を定義します。

❹描画する文字のフォントパターンを配列 pattern［ ］に格納します。そのとき，文字のタイプを変数 type に代入します。

type＝0	1/4 角
type＝1	半角
type＝2	全角

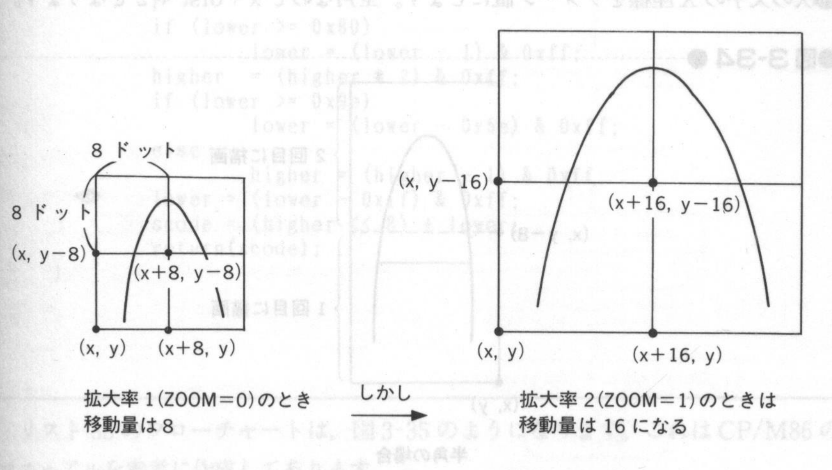
❺漢字を描画する場合 4 回に分けて行いますが，そのときの移動量を変数 ofst に入れます(図 3-32)。拡大率によってその値は変化します。

❻傾斜させるときには，X 方向の移動量がさらに違ってきます(図 3-33)。その値を変数 sofst に代入します。

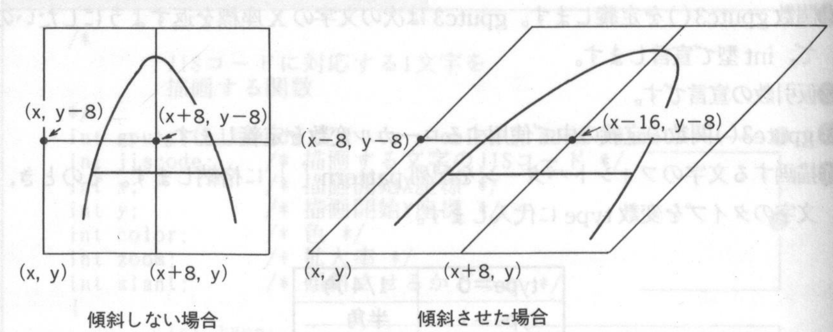
❼まず 1/4 角の場合を処理します。

❽ 1/4 角では描画が一度だけで済みます。

●図 3-32 ●

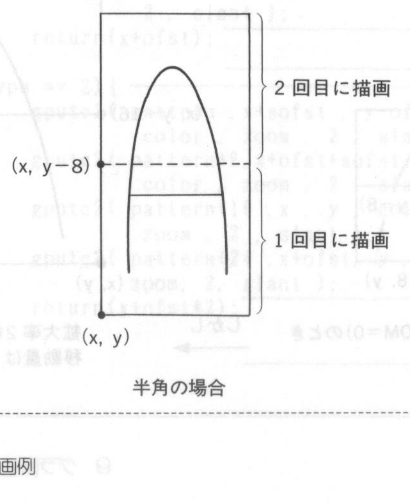


●図 3-33●



❾そして，次の文字を描画するX座標の値をリターン値にします。
❿半角の場合です。
⓫半角はY座標を変えて2回描画を行います(図3-34)。
⓬2回目に描画するフォントパターンは，配列の8バイト目以降に格納されています。
⓭次の文字を描画すべきX座標を返します。
⓮最後に全角の場合です。
⓯X座標，Y座標を変えながら4回描画しています。
⓰次の文字のX座標をリターン値にします。全角なのでx+ofst＊2となります。

●図 3-34●



gput3()関数はJISコードを引数に与えます。これはBIOSがJISコードしか受け付けないからです。ところが，MS-DOSで一般的に使われているのはシフトJISコードです。そのため，シフトJISコードからJISコードへの変換が必要です。この変換を行う関数jiscode()はリスト33のようになります。

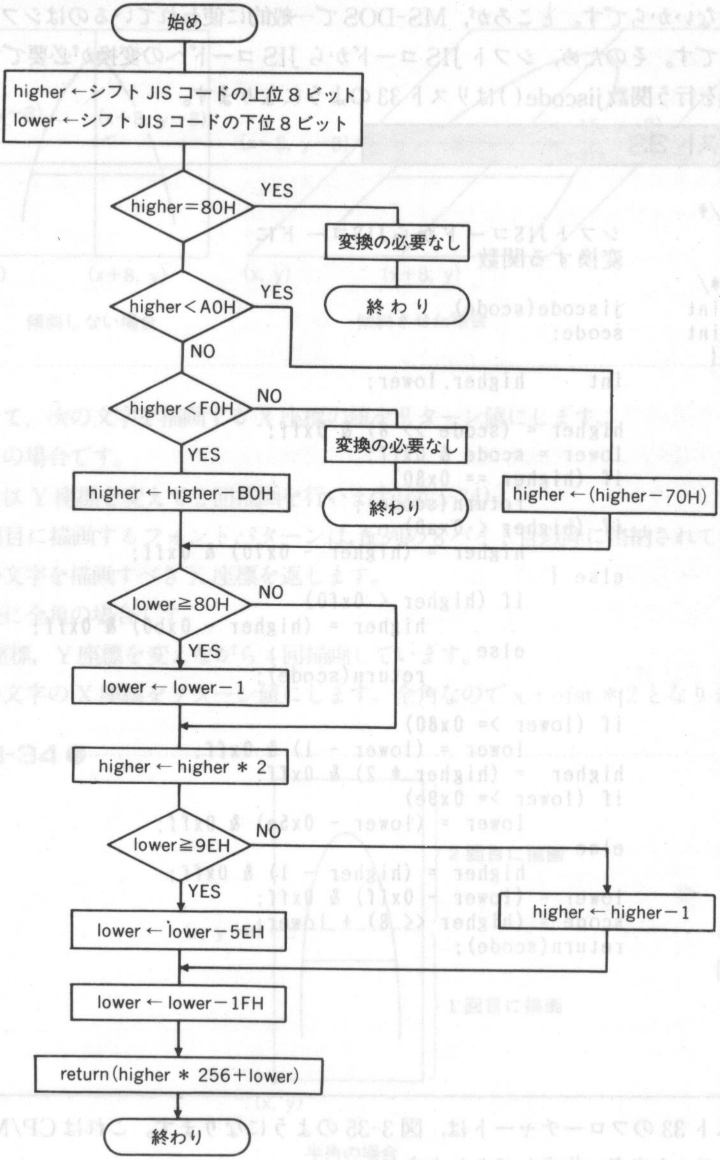
●リスト33

```
/*
        シフトJISコードからJISコードに
        変換する関数
*/
int     jiscode(scode)
int     scode;
{
        int     higher, lower;

        higher = (scode >> 8) & 0xff;
        lower = scode & 0xff;
        if (higher == 0x80)
                return(scode);
        if (higher < 0xa0)
                higher = (higher - 0x70) & 0xff;
        else {
                if (higher < 0xf0)
                        higher = (higher - 0xb0) & 0xff;
                else
                        return(scode);
        }
        if (lower >= 0x80)
                lower = (lower - 1) & 0xff;
        higher  = (higher * 2) & 0xff;
        if (lower >= 0x9e)
                lower = (lower - 0x5e) & 0xff;
        else
                higher = (higher - 1) & 0xff;
        lower = (lower - 0x1f) & 0xff;
        scode = (higher << 8) + lower;
        return(scode);
}
```

リスト33のフローチャートは，図3-35のようになります。これはCP/M86のマニュアルを参考に作成してあります。

●図 3-35●



始め
higher ←シフト JIS コードの上位 8 ビット
lower ←シフト JIS コードの下位 8 ビット
higher＝80H
YES
変換の必要なし
終わり
higher＜A0H
YES
NO
higher＜F0H
NO
変換の必要なし
YES
higher ← higher－B0H
終わり
higher ←(higher－70H)
lower≧80H
NO
YES
lower ← lower－1
higher ← higher ＊ 2
lower≧9EH
NO
YES
higher ← higher－1
lower ← lower－5EH
lower ← lower－1FH
return(higher ＊ 256＋lower)
終わり

gputc3(), jiscode()関数を使えば，この節の目的であるjprintf()関数は次のように定義できます(リスト34)。

◉リスト 34

```
/*
  日本語の文字列を描画する関数(拡大・傾斜可能)
*/
void jprintf(string,x,y,color,zoom,slant) ―――― ❶
char *string;      /* 文字列へのポインタ */
int x;             /* 描画開始X座標 */
int y;             /* 描画開始Y座標 */                ❷
int color;         /* 色 */
int zoom;          /* 拡大率 */
int slant;         /* 傾斜させるかどうか */
{
        int i;                                         ❸
        int code;

        for(i=0;i<=strlen( string )-1;i++){ ―――― ❹
                code =string[i]&0xff; ―――――――― ❺
                if  (((0x80<=code) && (code<=0x9f))
                  ||((0xe0<=code)&&(code<=0xff))){    ❻
                   code = code*256+(string[++i]&0xff);
                   code = jiscode( code );             ❼
                }
                x=gputc3( code,x,y,color,zoom,slant);  ❽
        }
}
```

解説

❶jprintf()関数を宣言します。

❷jprintf()関数の仮引数の宣言です。

❸jprintf()関数内で使用するローカル変数を宣言します。

❹文字列から１文字ずつ切り出してきます。

❺変数codeに文字を代入します。string［］がchar型なので，0xffとアンドをとっています。

❻もし，codeが漢字の前半分ならば，あとのコードをさらに読み込みます。

❼codeには２バイトのシフトJISコードがセットされているので，これを前出のjiscode()関数でJISコードに変換します。

❽変数 code には ASCII コードかシフト JIS コードが設定されました。それを gputc3()関数で表示します。gputc3()関数は次の文字の X 座標を返すので，変数 x にそれをセットします。

jprintf()関数が定義できました。jprintf()関数は文字が右端まで行くと左側から表示されます。また，Y 座標がはみ出す場合の処理は行っていません。必要ならば呼び出す前にクリッピング処理を行ってください。jprintf()関数をテストするプログラムを次に示します。〝日本ソフトバンク〟という文字列を拡大/傾斜させながら 8 回描画します(リスト 35)。

◉リスト 35

```
/*
        jprintf()関数のテスト
*/
void main()
{
        start();
        al();
        cls2();

        jprintf("日本ソフトバンク",0,15,1,0,FALSE);
        jprintf("日本ソフトバンク",0,35,2,0,TRUE);
        jprintf("日本ソフトバンク",0,75,3,1,FALSE);
        jprintf("日本ソフトバンク",0,115,4,1,TRUE);
        jprintf("日本ソフトバンク",0,175,5,2,FALSE);
        jprintf("日本ソフトバンク",0,235,6,2,TRUE);
        jprintf("日本ソフトバンク",0,315,2,3,FALSE);
        jprintf("日本ソフトバンク",0,395,7,3,TRUE);
}
```

1 gtype.c

gputc3()関数を利用した実用例として，gtype プログラムを示します。gtype は標準入力からのデータをグラフィック画面に表示します。たとえば

- ·type readme.doc | gtype

とすれば，readme.docファイルをグラフィック画面に表示します。起動時にオプションで色/文字サイズ/傾斜させるかどうかなどを変えることができます。括弧内はデフォルト値です。

-c	文字の色。0～7が指定できる。たとえば-c2ならば赤になる(-c7)
-z	文字のサイズ。0～15が指定可能。-z1ならば2倍の32×32ドットで漢字が表示される(-z0)
-s	文字を傾斜させるオプション(傾斜せず)
-h	起動時のオプションを表示させる。次のようなヘルプ表示がされる(表示せず) Usage：gtype [-c<n>] [-z<n>] [-s] [-?] [-h]
-?	起動時のオプションを表示させる(表示せず)

グラフィック画面いっぱいに表示されるとスクロールします。 gtype.cはリスト36のようになります。

◉リスト36

```
/*
  gtype.c
*/
#include <stdio.h> ──────────────────── ❶
#define PSET 0      ┐
#define PRESET 1    │
#define COMPLEMENT 2│ ❷
#define TRUE -1     │
#define FALSE 0     ┘

void main(argc,argv) ┐
int argc;            │ ❸
char *argv[];        ┘
{
  int c,c2;          ┐
  int i;             │
  int color=7;       │
  int zoom=0;        │ ❹
  int x=0;           │
  int y;             │
  int slant=FALSE;   │
  char *s;           ┘
```

```
    for(i=1;i<=argc-1;i++){ ─────────────────── ⑤
        s=argv[i]; ─────────────────────────── ⑥
        if(*s != '-'){ ─────────────────────── ⑦
            printf("illegal optionYn"); ───┐
            exit(1);                          ⑧
        } ─────────────────────────────────┘
        switch(*(s+1)){ ───────────────────── ⑨
        case 'c': ─────────────────────────── ⑩
            color=atoi( (s+2) ); ──────────── ⑪
            break; ─────────────────────────── ⑫
        case 'z': ─────────────────────────── ⑬
            zoom=atoi( (s+2) ); ───────────── ⑭
            break; ─────────────────────────── ⑮
        case 's': ─────────────────────────── ⑯
            slant=TRUE; ───────────────────── ⑰
            break; ─────────────────────────── ⑱
        case '?': ─────────────────────────── ⑲
            printf("Usage : gtype [-c<n>] [-z<n>] [-s] [-?]
            exit(0);                        [-h]Yn"); ⑳
        case 'h':
            printf("Usage : gtype [-c<n>] [-z<n>] [-s] [-?]
            exit(0);                        [-h]Yn"); ㉑
        default: ──────────────────────────┐
            printf("illegal optionYn");       ㉒
            printf("Usage : gtype [-c<n>] [-z<n>] [-s] [-?]
            exit(0);                        [-h]Yn");
        }
    }

    y=16*(zoom+1); ─────────────────────────── ㉓

    start(); ──────────────────────────────┐
    all();                                    ㉔
    cls2(); ───────────────────────────────┘
    while( (c = getchar()) != EOF){ ────────── ㉕
        switch( c ){ ───────────────────────── ㉖
        case 'Yn': ─────────────────────────── ㉗
            x=0; ───────────────────────────── ㉘
            y = y + 16*(zoom+1); ───────────── ㉙
            if( y>=400 ){ ──────────────────── ㉚
                y = y - 16*(zoom+1); ───────── ㉛
                scroll( (zoom+1)*16+1 ); ───── ㉜
            }
            break; ─────────────────────────── ㉝
```

```
        default:                                             ----(c<=0xff))){ ㉟
                    if(((0x80<=c) && (c<=0x9f))||((0xe0<=c)&&-
                            c2 = getchar();
                            c =(0xff &  c)*256+(0xff &c2);                    ㊱
                            c =  jiscode( c );
                                                                              ㊲
                    }
                    if ( c<= 0x20 ) c=0x20;                                   ㊳
                    if( x>= 640-16*(zoom+1)*(1-slant) ){                      ㊴
                            x=0;                                              ㊵
                            y = y + 16*(zoom+1);                              ㊶
                            if( y>=400 ){                                     ㊷
                                     y = y - 16*(zoom+1);                     ㊸
                                     scroll( (zoom+1)*16+1 );
                            }                                                 ㊹
                    }
                    x=gputc3( c,x,y,color,zoom,slant);
                    break;
                                                                              ㊺
            }
      }
}
```

解説

❶必要なヘッダファイルをインクルードします。

❷定数をマクロ定義します。

❸メインプログラムです。引数が指定できるようにmain()に仮引数を付けています。

❹main()関数内で使用するローカル変数の宣言です。

❺まず，起動時のオプションの処理を行います。

❻変数sにオプション文字列へのポインタを代入します。

❼たとえば

　A:¥>gtype -c7 -z1-s

で起動された場合，変数sには最初"-c7"へのポインタが代入されます。sの指す文字列の先頭1文字が"-"でない場合です。

❽不正なオプションなので，エラーメッセージを出力してプログラムを終了します。

❾それ以外の場合，sの指す文字列の2番目の値で場合分けをして処理します。switch文を使っています。2番目の文字は*(s+1)で与えられることに注意してください。

❿'c'の場合です。

⑪文字色の指定なので，3番目以下を色コードを表す文字列とみなし，atoi関数で数値化して変数colorに代入します。

⑫そしてswitch文を抜け出します。

⑬'z'の場合です。

⑭文字のサイズ(すなわち拡大率)の指定なので，変数zoomに指定した値を設定します。

⑮そしてswitch文を抜け出します。

⑯'s'の場合です。

⑰文字を傾斜させる指定なので，変数slantの値をTRUEにします。

⑱そしてswitch文を抜け出します。

⑲'?'の場合です。

⑳起動時のオプションを表示させプログラムを終了します。

㉑'h'の場合も同様です。

㉒それ以外の場合，定義されていないオプションなので，〝Illegal option〟を表示し起動時のオプションを表示してプログラムを終了します。

㉓以下はプログラム本体です。まず，拡大率に応じて文字を書き始めるY座標を調整します。

㉔グラフィック画面の初期化を行います。表示を開始させ，640×400モードにして画面を消去します。

㉕標準入力からの入力文字を変数cに代入します。そして，それがEOFでない間処理を続行します。

㉖文字の値によって，場合分けをします。

㉗入力文字が'¥n'のとき

㉘キャリッジリターンとラインフィードの動作をします。X座標を0にします。

㉙Y座標を1文字分進めます。

㉚Y座標が画面からはみ出す場合です。

㉛Y座標を元に戻します。

㉜1文字分スクロールさせます。関数scroll()はあとで定義します。スクロールさせる量は拡大率に応じて変えています。

㉝そしてswitch文を抜け出します。

㉞入力文字が'¥n'以外のときです。

㉟2バイトのシフトJISコードの場合です。
㊱もう1文字読み出して2バイトのコードにします。
㊲そして，そのシフトJISコードをJISコードに変換します。
㊳もし，文字が20H以下の表示できないものだったら，スペースに置き換えます。
㊴表示するX座標が画面の右からはみ出す場合です。
㊵キャリッジリターン，ラインフィードの処理を行います。まずX=0にします。
㊶そしてY座標を1文字分進めます。
㊷Y座標が画面からはみ出す場合です。
㊸Y座標をもとに戻します。
㊹画面を1文字分スクロールアップさせます。
㊺最後にgputc3()関数を使って文字をグラフィックス画面に表示させます。

scroll()関数の定義は，次のようになります(リスト37)。

●リスト37

```
/*
        グラフィックス画面をスクロール
        させる関数
*/
void scroll( dy )                                              ❶
unsigned int dy;                                               ❷
{
        long blue =0xa8000L;
        long red  =0xb0000L;
        long green=0xb8000L;                                   ❸

        unsigned int temp,test;
        int i,j;
        temp=dy*80U;                                           ❹

        while( (inp(0xa0) & 4) == 0 );                         ❺
        test=temp*temp;
        while( (inp(0xa0) & 8) != 0 );

        for(i=1;i<=10;i++) for(j=1;j<=1000;j++) test=temp*temp;

        memmove( (char *)blue ,(char *)(blue +temp),0x8000U-temp);❻
        memmove( (char *)red  ,(char *)(red  +temp),0x8000U-temp);❼
        memmove( (char *)green,(char *)(green+temp),0x8000U-temp);❽
```

```
    memset( (char *)(blue +0x8000L-temp),0x00,temp); ———— ❾
    memset( (char *)(red  +0x8000L-temp),0x00,temp); ———— ❿
    memset( (char *)(green+0x8000L-temp),0x00,temp);
}                                                    ———— ⓫
```

解説

❶まず，関数 scroll( )を宣言します。

❷仮引数はスクロールさせるドット数です。

❸以下ローカル変数です。各 VRAM の先頭アドレスを変数 blue，red，green に代入しています。

❹ dy ドットスクロールさせるには，VRAM の先頭から dy ＊ 80 バイト離れたデータを VRAM の先頭にブロック転送します(図 3-36)。このオフセットを変数 temp に入れます。

❺転送は CPU で行います。

❻青 VRAM のデータを転送します。

❼赤 VRAM のデータを転送します。

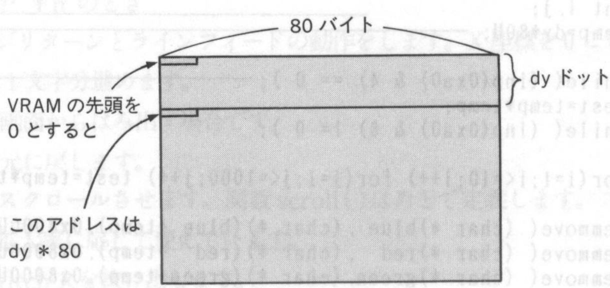
❽緑 VRAM のデータを転送します。

❾青 VRAM の最下行を 0 で消去します。

❿赤 VRAM の最下行を 0 で消去します。

⓫緑 VRAM の最下行を 0 で消去します。

●図 3-36 ●



プログラム gtype は標準入力からデータを入力するようになっているので，いろいろな使い方ができると思います。

```
A:¥>GTYPE
```

で起動すれば，キーボードから表示する文字列を入力することができます。

```
A:¥>TYPE README.DOC | GTYPE
```

のようにすれば，README.DOC ファイルの内容を表示することができます。

また

```
A:¥>DIR | GTYPE
```

とすれば，ディレクトリの内容をグラフィックス表示します。

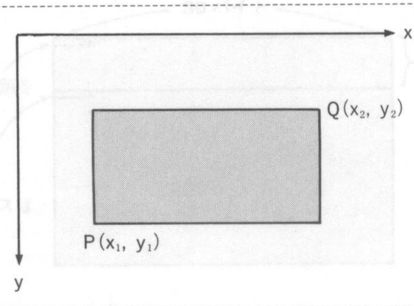
10 8×8ドット以外のグラフィックス文字描画

GDCは8×8ドット以外のグラフィックス文字描画も行えます。しかしその場合も，フォントパターンとしてGDCに送出できるのは，8×8ドットまでなので，それ以上大きな描画を指示しても8×8ドットのパターンの単なる繰り返しとなってしまいます。しかし，画面クリアやフィルなどの用途には役に立つでしょう。ここではボックスフィル(四辺形の内部を塗りつぶすこと)に使ってみます。ここで定義するのはbox_fill()関数です(図3-37)。

```
/*
        ボックスフィル
*/
void box_fill(x1,y1,x2,y2,color)
int x1;          /* 頂点PのX座標 */
int y1;          /* 頂点PのY座標 */
int x2;          /* 頂点QのX座標 */
int y2;          /* 頂点QのY座標 */
int color;       /* 塗りつぶす色 */
```

まずVRAM1プレーンに対して，ボックスフィルを行う関数box_fill0()を定義します。

●図3-37●



```
void box_fill0(x1,y1,x2,y2,plane,flag)
int x1;          /* 頂点PのX座標 */
int y1;          /* 頂点PのY座標 */
int x2;          /* 頂点QのX座標 */
int y2;          /* 頂点QのY座標 */
int plane;       /* 描画するVRAMのプレーン番号 */
int flag;        /* 描画するか消去するかのフラグ */
```

「8×8ドット以外のグラフィックス文字描画」を行うために送出する必要のあるコマンドは，次のとおりです。

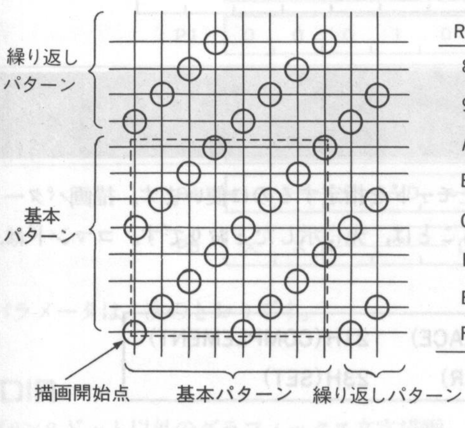
- TEXTWコマンド
- WRITEコマンド
- CSRWコマンド
- VECTWコマンド
- TEXTEコマンド

1 TEXTWコマンド

繰り返して描画するパターンを与えます(図3-38)。

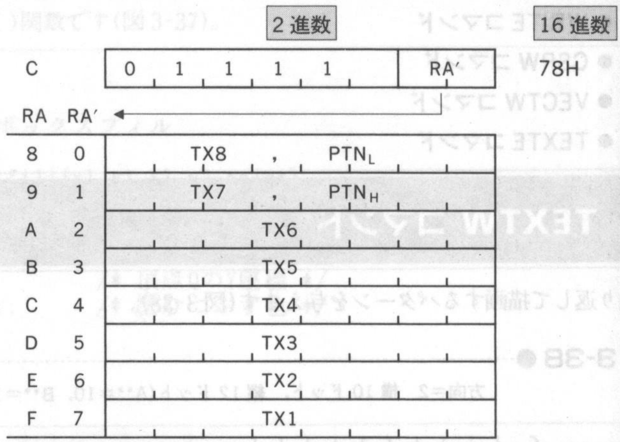
●図3-38●

方向=2　横10ドット，縦12ドット(A**=10，B**=11)



RA									
8	0	0	0	1	0	0	0	1	TX8
9	0	0	1	0	0	0	1	0	TX7
A	0	1	0	0	0	1	0	0	TX6
B	1	0	0	0	1	0	0	0	TX5
C	0	0	0	1	0	0	0	1	TX4
D	0	0	1	0	0	0	1	0	TX3
E	0	1	0	0	0	1	0	0	TX2
F	1	0	0	0	1	0	0	0	TX1
	LSB							MSB	

8×8 ドットのパターンの単純な繰り返し描画なので，漢字などを表示することはできません。また FFH, FFH, FFH, FFH, FFH, FFH, FFH, FFH は SET モードで描画すればボックスフィルになりますが，RESET モードで描画すると画面クリアになります。逆に 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H は SET モードで描画すれば画面クリアになり，RESET モードで描画するとボックスフィルになります。設定した描画パターンとドット修正モードに気を付けてください。ここでは FFH, FFH, FFH, FFH, FFH, FFH, FFH, FFH を SET モードで描画することでボックスフィルを実現しています。TEXTW のコマンドとパラメータは，次のとおりです。



2 WRITE コマンド

WRITE コマンドはドット修正モードを指定するのに使います。描画パターンとドット修正モードが関連していることは，先に示したとおりです。コマンドは，次のようになります。

コマンド	20H(REPLACE)	21H(COMPLEMENT)
	22H(CLEAR)	23H(SET)

付随するパラメータはありません。

3 CSRW コマンド

CSRW コマンドは描画を開始する点のアドレスを指定するのに使います。ほかの描画と同様にして EAD/dAD を計算してください。送出するコマンド/パラメータも同様です。

C	0	1	0	0	1	0	0	1
P1	EAD_L							
P2	EAD_M							
P3	dAD				0	0	EAD_H	

4 VECTW コマンド

描画方向，描画ドット数などを指定します。送出するコマンド/パラメータは，次のようになります。

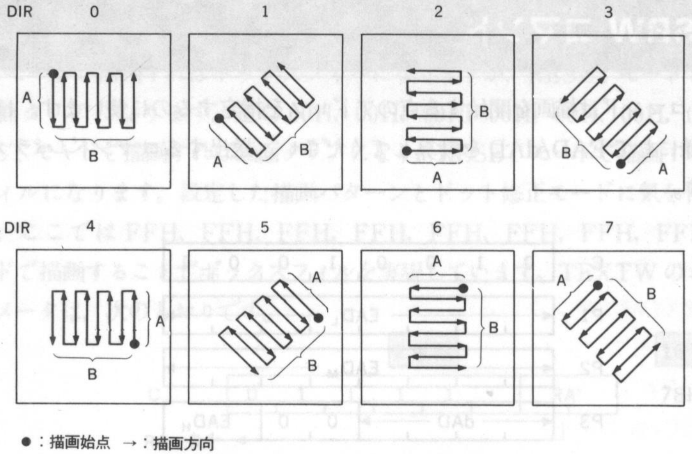
C	0	1	0	0	1	1	0	0
P1	0	0	0	1	0	DIR		
P2	DC_L							
P3	×	DGD	DC_H					
P4	D_L							
P5	×	×	D_H					

パラメータは，次のとおりです。

■ DIR

「8×8 ドット以外のグラフィックス文字描画」の描画方向を指定します（図 3-39）。

●図 3-39●



描画パラメータレジスタには，次の値を設定します。

■ DC レジスタ

第2描画方向ドット数を指定します。第2描画ドット数は，図3-39のBに相当します。

■ D レジスタ

第1描画方向ドット数を指定します。第1描画ドット数は，図3-39のAに相当します。

■ D2 レジスタ

設定禁止

■ D1 レジスタ

設定禁止

■ DM レジスタ

設定禁止

「8×8ドット以外のグラフィックス文字描画」では，VECTWで以上のように，5つのパラメータを送出します。

5 TEXTEコマンド

「8×8ドット以外のグラフィックス文字描画」を開始させます。グラフィックス文字関係の描画の場合は，VECTEではなくTEXTEコマンドを使います。TEXTEは

コマンド	12H
パラメータ	なし

です。

これらのコマンドを使うと，box_fill0()関数はリスト38のようになります。

●リスト38

```
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2                                    ❶

char fill_pattern[8]={0xff,
                      0xff,
                      0xff,
                      0xff,
                      0xff,
                      0xff,
                      0xff,
                      0xff};                            ❷

void box_fill0(x1,y1,x2,y2,plane,flag)                  ❸
int x1;          /* 頂点PのX座標 */
int y1;          /* 頂点PのY座標 */
int x2;          /* 頂点QのX座標 */
int y2;          /* 頂点QのY座標 */                      ❹
int plane;       /* 描画するVRAMのプレーン番号 */
int flag;        /* 描画するか消去するかのフラグ */
{
        unsigned ead,ead_offset;
        unsigned dad;
        int i;
```

```
int dc_reg;        /* DC  レジスタ */
int d_reg;         /* D    レジスタ */

int dx;
int dy;
int temp;

dx = abs(x2-x1);
dy = abs(y2-y1);
if( y2>y1 ){
        temp=y1;
        y1=y2;
        y2=temp;
}

if( x2<x1 ){
        temp=x1;
        x1=x2;
        x2=temp;
}

/* textw */
send_command( 0x78 );   /* textw command */
for(i=0;i<=7;i++)
        send_parameter( fill_pattern[i] );

/* write */
if( flag == PSET )
        send_command( 0x23 );
if( flag == PRESET )
        send_command( 0x22 );
if( flag == COMPLEMENT )
        send_command( 0x21 );

/* culadr */

ead_offset=0x4000U+0x4000U*plane;
ead = ead_offset+40*y1+x1/16;
dad = x1 % 16;

/* csrw */
send_command( 0x49 );
send_word_parameter( ead );
send_parameter( 16*dad );

dc_reg = dy;
d_reg = dx;
```

❺ ❻ ❼ ❽ ❾ ❿ ⓫ ⓬ ⓭ ⓮ ⓯ ⓰ ⓱

```
        /* vectw */
        send_command( 0x4c ); ──────────────────── ⑱
        send_parameter( 0x12 );      /* dir = 2 */── ⑲
        send_word_parameter( dc_reg ); ─────────┐
        send_word_parameter( d_reg ); ──────────┘ ⑳

        /* tecte */
        send_command( 0x68 ); ─────────────────── ㉑
}
```

解説

❶定数をマクロ定義します。

❷ボックスフィル時に書き込みを繰り返すパターンを配列 fill_pattern[8] に設定します。ほかの関数から変更できるように外部配列にしています。

❸box_fill0()関数の宣言部です。

❹仮引数です。

❺box_fill0()関数内で使用するローカル変数を宣言します。

❻まず，X座標の差の絶対値を dx に代入します。

❼次に，Y座標の差の絶対値を dy に代入します。

❽図 3-37 で点 $P(x_1, y_1)$ が四辺形の右下，点 $Q(x_2, y_2)$ が右上になるように座標値を正規化します。$y_2 > y_1$ ならば y_1 と y_2 とを交換します。

❾同様に，$x_2 < x_1$ ならば x_1 と x_2 とを交換します。

❿次に，TEXTW コマンドを送出します。コマンドは 78H です。

⓫配列 fill_pattern［ ］に入っているパターンを GDC にパラメータ送出します。

⓬ WRITE コマンドを送出しドット修正モードを選択します。flag の値に従って 21H か 22H か 23H をコマンド送出します。

⓭描画開始アドレスを計算します。点Pが必ず右下になっているため，x_1，y_1 から描画を開始します。点 (x_1, y_1) に対応するアドレス，ドットアドレスを計算します。

⓮ CSRW コマンドを送出し，描画開始アドレスを GDC に指示します。CSRW コマンドは 49H です。

⑮EAD, dADをパラメータ送出します。
⑯第1描画方向ドット数をdc_regに代入します。描画はすべてdir=2で行うので図3-39よりdc_reg=dyになることがわかります。
⑰同様にDレジスタの値を変数d_regに代入します。d_reg=dxです。
⑱これらの描画パラメータレジスタの値をGDCに知らせるためにVECTWコマンドを送出します。VECTWコマンドは4CHです。
⑲描画方向は2としているので12Hをパラメータ送出します。
⑳dc_reg, d_regをパラメータ送出します。
㉑TEXTEコマンドを送りGDCに「8×8ドット以外のグラフィックス文字描画」を開始させます。TEXTEコマンドは68Hです。

これでVRAM1プレーンに対してボックスフィルを行う関数box_fill0()が定義できました。これを使って目的のbox_fill()関数を定義します。box_fill()関数はリスト39のようになります。

◉リスト39

```
/*
        ボックスフィル
*/
void box_fill(x1,y1,x2,y2,color)
int x1;            /* 頂点PのX座標 */
int y1;            /* 頂点PのY座標 */
int x2;            /* 頂点QのX座標 */
int y2;            /* 頂点QのY座標 */
int color;         /* ぬり潰す色 */
{
        if( (color & 1) != 0 )
                box_fill0(x1,y1,x2,y2,0,PSET);
        else
                box_fill0(x1,y1,x2,y2,0,PRESET);
        if( (color & 2) != 0 )
                box_fill0(x1,y1,x2,y2,1,PSET);
        else
                box_fill0(x1,y1,x2,y2,1,PRESET);
        if( (color & 4) != 0 )
                box_fill0(x1,y1,x2,y2,2,PSET);
        else
                box_fill0(x1,y1,x2,y2,2,PRESET);
}
```

R, G, B3プレーンに対して描画/消去を繰り返してボックスフィルを行っています。リスト 40 に，box _ fill( ) 関数を使ったデモを示します。色を変えながら画面全体のボックスフィルを行うものです。

◉リスト 40

```
/*
        ボックスフィルのテスト
*/
void main()
{
        int i;
        int x1,y1;
        int x2,y2;
        int color=0;

        start();
        all();
        cls2();
        for(i=300;i>=3;i -= 20){
                x1 = 320 - i;
                y1 = 200 - i*400L/600;
                x2 = 320 + i;
                y2 = 200 + i*400L/600;
                color = color % 7+1;
                box_fill( x1 , y1 , x2 , y2 , color );
        }
}
```

塗りつぶしで使うデータは外部配列にしているので，プログラムの中で変更する

●図 3-40●



ことができます。図3-40に示すような三角形で塗りつぶしたければ配列の内容は

```
char fill_pattern[8]={  0x80,   /* 10000000B */
                        0xc0,   /* 11000000B */
                        0xe0,   /* 11100000B */
                        0xf0,   /* 11110000B */
                        0xf8,   /* 11111000B */
                        0xfc,   /* 11111100B */
                        0xfe,   /* 11111110B */
                        0xff};  /* 11111111B */
```

に変更します。リスト40を三角形で塗りつぶすように変更すると，リスト41のようになります。

◉リスト41

```
/*
    三角形のパターンでボックスフィル
  を行なう
 */
void main()
{
        int i;
        int x1,y1;
        int x2,y2;
        int color=0;

        char temp[8]={  0x80,   /* 10000000B */
                        0xc0,   /* 11000000B */
                        0xe0,   /* 11100000B */
                        0xf0,   /* 11110000B */
                        0xf8,   /* 11111000B */
                        0xfc,   /* 11111100B */
                        0xfe,   /* 11111110B */
                        0xff};  /* 11111111B */

        start();
        all();
        cls2();

        for(i=0;i<=7;i++)
                fill_pattern[i] = temp[i];

        for(i=300;i>=3;i -=20){
```

```
            x1 = 320 - i;
            y1 = 200 - i*400L/600;
            x2 = 320 + i;
            y2 = 200 + i*400L/600;
            color = color % 7+1;
            box_fill( x1 , y1 , x2 , y2 , color );
        }
}
```

また，box _ fill( ) 関数を使えば，グラフィックス画面を全部消去するコマンドを簡単に作ることができます。画面消去のプログラムをリスト42に示します。

◉リスト 42

```
/*
        グラフィックス画面の消去
*/
void main()
{
        box_fill(0,0,639,399,0);
}
```

また，box _ fill0( ) 関数を使うと画面を反転させることができます。写真のネガとポジが反転するような感じになります。画面反転の例リスト43は，ボックスを20個描いてから画面反転し，もう一度反転することでもとに戻しています。

◉リスト 43

```
/*
        画面反転
*/
#include <stdlib.h>

#define PSET 0
#define PRESET 1
#define COMPLEMENT 2

void main()
{
        int i,c;
        int x1,y1;
        int x2,y2;
        int color;
```

```
        for(i=1;i<=20;i++){

                x1 = rand() % 640;
                y1 = rand() % 400;
                x2 = rand() % 640;
                y2 = rand() % 400;
                color = rand() % 7+1;

                box_fill(x1,y1,x2,y2,color);
        }

        printf("画面を反転します。");
        printf("リターンキーを押してください。");

        dummy = getchar();

        box_fill0(0,0,639,399,0,COMPLEMENT)
        box_fill0(0,0,639,399,1,COMPLEMENT)
        box_fill0(0,0,639,399,2,COMPLEMENT)

        printf("画面を元に戻します。");
        printf("リターンキーを押してください。");

        dummy = getchar();

        box_fill0(0,0,639,399,0,COMPLEMENT)
        box_fill0(0,0,639,399,1,COMPLEMENT)
        box_fill0(0,0,639,399,2,COMPLEMENT)
}
```

なお，これらのプログラムを実行すればわかりますが，GDCのボックスフィルはCPUに比べてかなり遅いものです。速度が重要な場合は，CPUでボックスフィルを行うプログラムを書くべきでしょう。

11 傾斜した四辺形の塗りつぶし

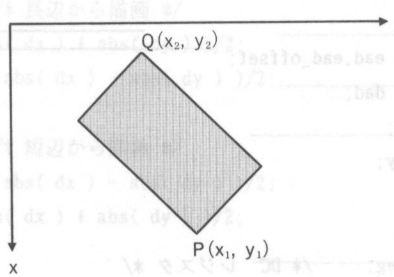
「8×8ドット以外のグラフィックス文字描画」を使うと，傾斜した四辺形の塗りつぶしが行えます。方法は，描画方向を斜めにしてグラフィックス文字描画を行うだけです。

図3-41のような傾斜四辺形を塗りつぶす関数box_ fill2( )を，次のように定義します。

```
/*
        傾斜した四辺形の描画
*/
void box_fill2(x1,y1,x2,y2,color)
int x1;          /* 頂点PのX座標 */
int y1;          /* 頂点PのY座標 */
int x2;          /* 頂点QのX座標 */
int y2;          /* 頂点QのY座標 */
int color;       /* 塗りつぶす色 */
```

まず，VRAM1プレーンに対して描画する関数box_ fill20( )を定義します。box_ fill20( )はリスト44のようになります。

●図3-41●



●リスト44

```
/*
        VRAM1プレーンに対して四辺形塗りつぶしを
        行なう関数
*/
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2                              ❶

char fill_pattern[8]={0xff,
                 0xff,
                 0xff,
                 0xff,
                 0xff,
                 0xff,
                 0xff,
                 0xff};                            ❷

void box_fill20(x1,y1,x2,y2,plane,flag)
int x1;
int y1;
int x2;
int y2;
int plane;
int flag;
{                                                  ❸
        unsigned ead,ead_offset;
        unsigned dad;
        int temp;
        int dx,dy;
        int a,b;                                   ❹

        int dc_reg;     /* DC  レジスタ */
```

```
    int d_reg;       /* D  レジスタ */

    int i;

    /* textw */
    send_command( 0x78 );   /* textw command */          ❺
    for(i=0;i<=7;i++)                                     ❻
        send_parameter( fill_pattern[i] );

/* write */
if( flag == PSET )
    send_command( 0x23 );
if( flag == PRESET )
    send_command( 0x22 );                                 ❼
if( flag == COMPLEMENT )
    send_command( 0x21 );

if( y1 < y2 ){
    temp = x1; x1 = x2; x2 = temp;                        ❽
    temp = y1; y1 = y2; y2 = temp;
}

dx = x2 - x1;                                             ❾
dy = y2 - y1;

if( x1 < x2 ){  /* 長辺から描画 */
    a = ( abs( dx ) + abs( dy ) )/2;                      ❿
    b = abs( abs( dx ) - abs( dy ) )/2;
}
if( x1 > x2){   /* 短辺から描画 */
    a = abs( abs( dx ) - abs( dy ) )/2;                   ⓫
    b = ( abs( dx ) + abs( dy ) )/2;
}
```

```
if( x1 == x2){
      a = ( abs( dx ) + abs( dy ) )/2;
      b = a;                                                    ⓬
}

      /* culadr */

      ead_offset=0x4000U+0x4000U*plane;
      ead = ead_offset+40*y1+x1/16;                             ⓭
      dad = x1 % 16;

      /* csrw */
      send_command( 0x49 );
      send_word_parameter( ead );                               ⓮
      send_parameter( 16*dad );

      /* culpar */
      dc_reg = b;                                               ⓯
      d_reg = a;

      /* vectw */
      send_command( 0x4c );
      send_parameter( 0x13 );           /* dir = 3 */
      send_word_parameter( dc_reg );                            ⓰
      send_word_parameter( d_reg );

      /* tecte */
      send_command( 0x68 );                                     ⓱
}
```

解説

❶定数を定義します。

❷次に塗りつぶすパターンを格納する配列 fill_ puttern［ ］を定義します。外部変数で宣言しているのは，どの関数からも塗りつぶしパターンを変更できるようにするためです。全部 FFH なので，べた塗りになります。

❸box_ fill20( )関数を宣言します。値を返さないので void 型で宣言しています。

❹ローカル変数の宣言です。

❺まず，塗りつぶしのパターンを GDC に知らせるために，TEXTW コマンド(78H)を送出します。

❻配列 fill_pattern［ ］の内容をパラメータ送出します。

❼WRITE コマンドを送出し，ドット修正モードを指定します。変数 flag の値によって送出コマンドを選択します。

❽座標値を正規化します。もし，P よりも Q のほうが画面の下側にある場合，座標値をすべて入れ替えます。

❾X 座標の増分を dx，Y 座標の増分を dy に入れます。

❿第 1 描画方向ドット数を a に，第 2 描画ドット数を b に代入します。$P(x_1, y_1)$ $Q(x_2, y_2)$ から a，b を計算するには，3 つの場合に分ける必要があります(図 3-42)。

まず，P が Q よりも右側にある場合

$a=(|dx|+|dy|)/2$

$b=||dx|-|dy||/2$

なる関係があります。したがってプログラムは，このようになります。

⓫次に，P が Q より左側にある場合は

$a=||dx|-|dy||/2$

$b=(|dx|+|dy|)/2$

という関係があります。プログラムは，このようになります。

⓬最後に P，Q の X 座標が一致する場合

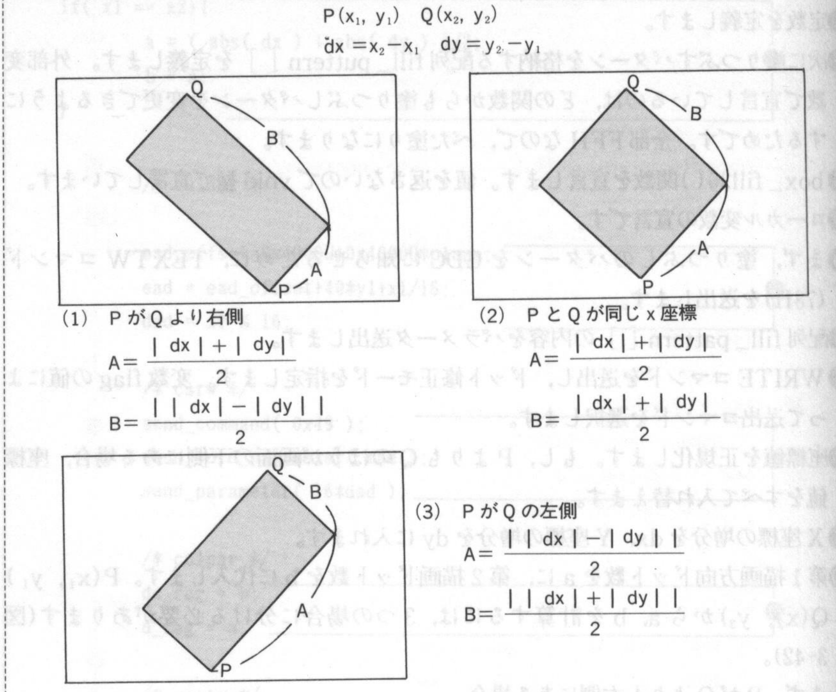
$a=(|dx|+|dy|)/2$

$b=(|dx|+|dy|)/2$

となります。

⓭次に，描画を開始するアドレス/ドットアドレスを計算します。

●図3-42●



⑭計算した開始アドレス，ドットアドレスをCSRWコマンドでGDCに知らせてやります。CSRWコマンドは49Hです。

⑮次に，VECTWコマンド用のパラメータを計算します。設定するのはDCレジスタ，Dレジスタです。DCレジスタには第2描画方向ドット数，Dレジスタには第1描画方向ドット数を与えます。順番が第2，第1なので注意してください。

⑯VECTWコマンドを送出します。VECTWコマンドは4CHです。描画方向は3です。

⑰パラメータはDCレジスタ，Dレジスタまでです。ほかのD2，D1，DMは設定しません。

最後に，TEXTEコマンド(68H)を送出し，塗りつぶしを開始します。

box_fill20()関数を3回呼び出して完全な四辺形塗りつぶしにするbox_fill2()関数はリスト45のようになります。

●リスト45

```
/*
        傾斜した四辺形塗りつぶしを行う
        関数
*/
void box_fill2( x1 , y1 , x2 , y2 , color )
int x1;         /*      左上のX座標     */
int y1;         /*      左上のY座標     */
int x2;         /*      右下のX座標     */
int y2;         /*      右下のY座標     */
int color;      /*      色              */
{
        if( (color & 1) != 0 )
                box_fill20(x1,y1,x2,y2,0,PSET);
        else
                box_fill20(x1,y1,x2,y2,0,PRESET);
        if( (color & 2) != 0 )
                box_fill20(x1,y1,x2,y2,1,PSET);
        else
                box_fill20(x1,y1,x2,y2,1,PRESET);
        if( (color & 4) != 0 )
                box_fill20(x1,y1,x2,y2,2,PSET);
        else
                box_fill20(x1,y1,x2,y2,2,PRESET);
}
```

変数colorの値に応じて描画消去を3回行っています。box_fill2()関数の使用例をリスト46に示します。片方の頂点を(320, 50)に固定し，もう一方の座標を(x, 350)にして，xと色を変えながら傾斜四辺形を塗りつぶしています。

◉リスト46

```
/*
          box__fill2()関数のテスト
*/
void main()
{
          int i,x,color;
          int x1,y1,x2,y2;
          start();
          all();
          cls2();

          for(i=0;i<=88;i++){
                    x = i*5+100;
                    color = i % 7 +1;
                    box_fill2( 320 , 50 , x , 350 , color);
          }
}
```

12 1ドット描画

GDCは1ドット描画の機能をもちます。これはグラフィックス画面に点を打つ機能です。1ドット描画は，PC-9801の場合，CPUで行うほうがはるかに高速です。ここではGDCで1ドット描画を行う関数pset( )を定義します。

```
void pset( x , y , color )
int x;          /*      点のX座標         */
int y;          /*      点のY座標         */
int color;      /*      色                */
```

1ドット描画を行うのに必要なコマンドは，次のとおりです。

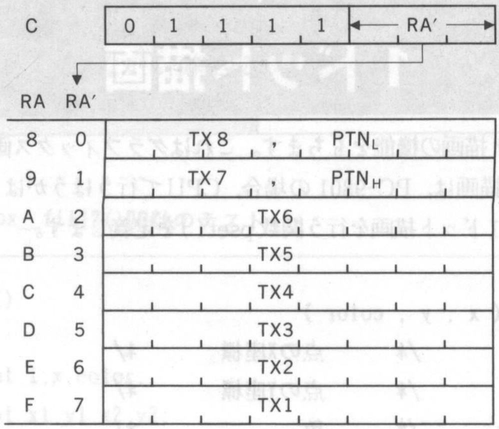
- TEXTWコマンド
- WRITEコマンド
- CSRWコマンド
- VECTWコマンド
- VECTEコマンド

1 TEXTWコマンド

「1ドット描画」でTEXTWコマンドの果たす役割は，点を打つかどうかということです。1ドットしか描画しないので，パラメータとして送出するのは0か1です。TEXTWのコマンド/パラメータは，次ページ上のとおりです。

2 WRITEコマンド

ドット修正モードを定義します。REPLACE, COMPLEMENT, CLEAR, SETのいずれかを選択します。「1ドット描画」の場合，REPLACEとSETの意味はまったく同じです。TEXTWで1を描画パターンに指定した場合，ドット修正



モードを SET にすれば点の描画，CLEAR にすれば点の消去となります。逆に 0 を指定した場合，SET，REPLACE にすれば点の消去になります。TEXTW のパターンとドット修正モードは関係があるので注意してください。WRITE のコマンド/パラメータは，次のとおりです。

コマンド	20H REPLACE モード時 21H COMPLEMENT モード時 22H CLEAR モード時 23H SET モード時
パラメータ	なし

3 CSRW コマンド

描画する点のアドレスを GDC に知らせるのが，CSRW コマンドの役割です。ほかと同様の計算式で描画開始アドレス，ドットアドレスを計算し GDC に送出します。CSRW のコマンド/パラメータは，次のとおりです。

C	0	1	0	0	1	0	0	1
P1	← EAD$_L$ →							
P2	← EAD$_M$ →							
P3	← dAD →				WG	0	EAD$_H$	

4 VECTW コマンド

描画種類，描画用パラメータレジスタの値を GDC に知らせます。コマンド/パラメータは，次のとおりです。

	2進数								16進数
C	0	1	0	0	1	1	0	0	4CH
P1	SL	R	C	T	L	← DIR →			
P2	← DC$_L$ →								
P3	×	DGD	← DC$_H$ →						
P4	← D$_L$ →								
P5	×	×	← D$_H$ →						
P6	← D2$_L$ →								
P7	×	×	← D2$_H$ →						
P8	← D1$_L$ →								
P9	×	×	← D1$_H$ →						
P10	← DM$_L$ →								
P11	×	×	← DM$_H$ →						

■ DIR

描画方向ですが，1ドット描画のときは意味がありません。しかし，設定する必要はあります。この節では0を指定しています。

■ DCレジスタ

設定禁止

■ Dレジスタ

設定禁止

■ D2レジスタ

設定禁止

■ D1レジスタ

設定禁止

■ DMレジスタ

設定禁止

このように「1ドット描画」の場合，DC, D, D2, D1, DMレジスタの値は設定禁止です。設定禁止というのは適当な値をパラメータ送出してよいという意味ではなく，対応するパラメータは送出してはならないということです。

5 VECTEコマンド

「1ドット描画」を開始させます。TECTEコマンドではない点に注意してください。VECTEコマンドは，次のようになります。

	2進数								16進数
コマンド	0	1	1	0	1	1	0	0	6CH

パラメータ　　なし

以上から，VRAM1プレーンに対して点を打つ関数pset0()は，リスト47のようになります。TEXTWのパラメータを1にして点を打つ場合はSETモードを,点を消去する場合はCLEARモードを，使用しています。

◉リスト47

```
/*
      VRAM1プレーンに対して点を描画する
      関数
*/
#define PSET 0
#define PRESET 1
#define COMPLEMENT 2                          ❶

void pset0(x,y,plane,flag)
int x;
int y;
int plane;
int flag;                                     ❷
{
      unsigned ead,ead_offset;
      unsigned dad;                           ❸

      /* textw */
      send_command( 0x78 );   /* textw command */
      send_word_parameter( 0x0001 );  /* pattern */   ❹

      /* write */
      if( flag == PSET )
            send_command( 0x23 );
      if( flag == PRESET )
            send_command( 0x22 );
      if( flag == COMPLEMENT )
            send_command( 0x21 );             ❺
```

```
    /* culadr */

    ead_offset=0x4000U+0x4000U*plane;
    ead = ead_offset+40*y+x/16;                    ❻
    dad = x % 16;

    /* csrw */
    send_command( 0x49 );
    send_word_parameter( ead );                    ❼
    send_parameter( 16*dad );

    /* vectw */
    send_command( 0x4c );                          ❽
    send_parameter( 0x00 );

    /* vecte */
    send_command( 0x6c );                          ❾
}
```

解説

❶プログラム中で使用する定数を定義します。

❷pset0()関数の定義です。値を返さないのでvoid型で宣言しています。

❸ローカル変数の宣言です。

❹TEXTWコマンドを送出します。コマンドは78Hです。パラメータは1を設定しています。

❺次にWRITEコマンドを送出してドット修正モードをGDCに知らせます。例によって変数フラグの値に応じてコマンドを変えています。

❻描画開始アドレスEAD，描画開始ドットアドレスdADを計算します。

❼CSRWコマンドで今計算したEAD，dADを送出します。CSRWコマンドは49Hです。

❽VECTWコマンドを送出します。描画方向はDIR=0としています。
❾VECTEコマンドを送出し「1ドット描画」を開始させます。

いま定義したpset0()関数を使えば，ここでの目的であるpset()関数はリスト48のように定義できます。

●リスト48

```
/*
      1ドット描画を行なう関数
*/
void pset( x , y , color )
int x;          /*      点のX座標        */
int y;          /*      点のY座標        */
int color;      /*      色               */
{
      if( (color & 1) != 0 )
            pset0( x,y,0,PSET);
      else
            pset0( x,y,0,PRESET);
      if( (color & 2) != 0 )
            pset0( x,y,1,PSET);
      else
            pset0( x,y,1,PRESET);
      if( (color & 4) != 0 )
            pset0( x,y,2,PSET);
```

pset0()関数を3回呼ぶことで1ドット描画を完成させています。pset()関数の使用例をリスト49に示します。ランダムな位置にランダムな色の点を3000個描画する例です。

●リスト 49

```
/*
        pset()関数の使用例
*/
void main()
{
        int i;
        int x,y;
        int color;

        start();
        all();
        cls2();
        for(i=1;i<=3000;i++){
                x = rand() % 640;
                y = rand() % 400;
                color = rand() % 7 +1;
                pset( x, y, color);
        }
}
```

13 VRAMからのデータ読み出し

これまで，GDCの描画機能について述べてきました。GDCは線画や塗りつぶしのほかに，データをVRAMから生の形で読み出す機能をもっています。せっかく描画したグラフィックス画面も保存できないと価値が半減するでしょう。レイトレーシングなどのように，絵を完成させるために何10時間とかかるプログラムなどでは，絵が保存できないと扱いに困ってしまいます。ここでは，PC-9801のVRAMの内容をGDC経由でディスクにセーブする関数gsave()を定義します。

```
void gsave(fname);
char *fname;      /* セーブするファイル名 */
```

さて，VRAMの内容を読み出す場合，送出するコマンドは，次のとおりです。

- CSRWコマンド
- MASKコマンド
- VECTWコマンド
- READコマンド

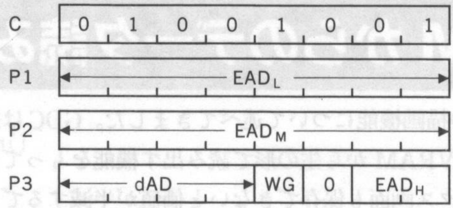
1 CSRWコマンド

VRAMの内容を読み出すとき，CSRWコマンドの果たす役割は，読み出しを開始するアドレスの指示です。GDCへは，読み出し開始アドレスのみを指示します。描画のようにドットアドレスは指示しません。CSRWのコマンドとパラメータは，次ページのとおりです。

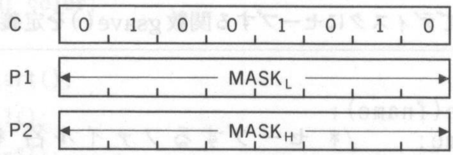
2 MASKコマンド

マスクレジスタの値を設定します。読み出しの場合，FFFFHを設定します。

CSRW コマンドとパラメータは，次のとおりです。



MASK コマンドとパラメータは，次のとおりです。



3 VECTW コマンド

VECTW コマンドは読み出しを行うバイト数(ワード数)の指定と，1データを読み出すごとに，アドレスをどう変化させるかを指示します。VECTW のコマンド/パラメータは，次ページ上のとおりです。

各パラメータは，次のように設定します。

■ DIR

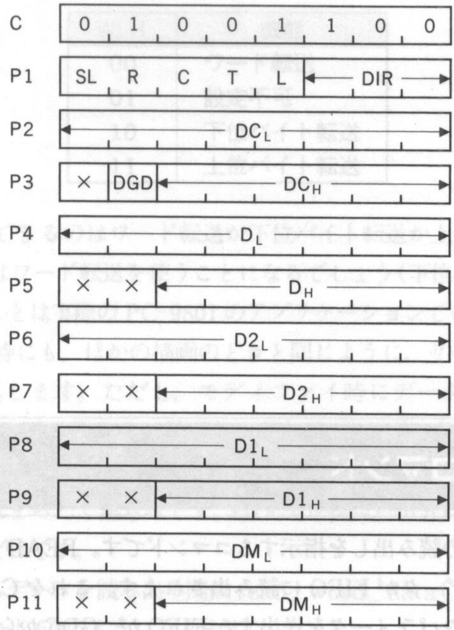
読み出しの方向を指定します(図 3-43)。

通常，DIR＝2 に設定しますが，斜めにデータを読み出すことも可能なわけです。

■ DC レジスタ

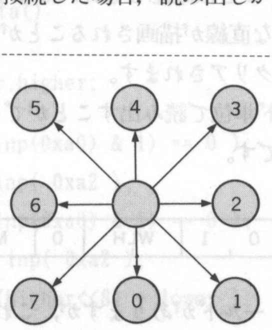
読み出すデータのバイト数(ワード数)を設定します。次ページに示したコマンド/パラメータの DC のフィールドを見ればわかるように，DC レジスタは 14 ビットのレジスタです。14 ビットで指示できる値は符号なしの場合

$0 \sim 2^{14}-1(=16383)$



です。つまり，GDCは一度に16383ワードしか読み出せないのです。これは，PC-9801でいえばVRAM1プレーン分です。これよりも広い領域を読み出す場合は，読み出しアドレスを変更して，16Kワード以内の読み出しを繰り返して行います。たくさんのVRAMを接続した場合，読み出しが多少繁雑になります。

●図3-43●



■ Dレジスタ

設定禁止

■ D2 レジスタ

設定禁止

■ D1 レジスタ

設定禁止

■ DM レジスタ

設定禁止

4 READ コマンド

実際に，データの読み出しを指示するコマンドです。READ コマンドが送出されると，VRAM のデータが FIFO に読み出されます。これを CPU で読み出します。いつもはコマンドやパラメータを送出する FIFO が，GDC から CPU の方向に変わって使用されるわけです。READ による読み出しを途中で終了する場合，FIFO に何かコマンドを送ってやります。すると，今まで GDC→CPU の方向だった FIFO が GDC←CPU の方向に変わります。このとき FIFO の状態を調べる必要はありません。仮に，FIFO がデータでいっぱいだとしても無条件にコマンドを送出します。しかし，GDC が描画中でないことは確認しなければなりません。もし，描画中にコマンドを送出すると，おかしな直線が描画されることがあります。なお，READ が実行されると FIFO の内容がクリアされます。

GDC はバイト単位とワード単位で読み出すことができます。READ のコマンドとパラメータは，次のとおりです。

C	1	0	1	WLH	0	MOD

コマンドの中に WLH フィールドがありますが，これでバイト転送かワード転送かなどを指定します。

WLH	機能
00	ワード転送
01	設定不可
10	下位バイト転送
11	上位バイト転送

このように指定できるのはワード転送か下位バイト転送か上位バイト転送かです。通常の読み出しではワード転送を使うことになるでしょう(下位バイトだけ，上位バイトだけ使用することは実際のPC-9801のアプリケーションでは稀だと思います)。

GDCはREAD時にも，ほかの描画のときと同じように，リードモディファイライト動作を繰り返します。ただし，モディファイ時にデータは変化しません。READコマンドにMODフィールドがありますが，この値に従ってREPLACE，SET，CLEAR，COMPLEMENTという動作をしたりはしないわけですから，MODフィールドはまったく無意味です。GDCの内部のMODレジスタの内容を単に書き換えているだけです。

まず，FIFOからデータを読み出す関数read_ word_ data()を定義します。

プログラムはリスト50のようになります。

●リスト50

```
/*
        FIFOからデータを読み出す関数
*/
int read_word_data()
{
        int lower,higher;

        while( (inp(0xa0) & 1) == 0 );
        lower = inp( 0xa2 );
        while( (inp(0xa0) & 1) == 0 );
        higher = inp( 0xa2 );
        return( (higher<<8) + lower );
}
```

解説

❶read_ word_ data( )を定義します。

❷read_ word_ data( )内で使用するローカル変数です。

❸GDCのステータスフラグ(I/OポートのA0H)を読み込み，DATA　READYかどうかを調べます。DATA　READYはステータスフラグのビット0です。DATA　READYの場合，VRAMのデータがFIFOにセットされていることを示すので，変数lowerに読み出します(データリードポート＝A2H)。

❹同様に上位バイトも読み出します。

❺変数higherを256倍し，lowerの値を加えて戻り値とします。

GDCは，一度に16383ワードしか読み出せないので，読み出しは3回に分けて行います。VRAM1プレーンに対して読み出しを行い，ファイルに書き込む関数gsave0( )を定義します。関数gsave0( )はリスト51のようになります。

◉リスト51

```
/*
        VRAM1プレーンをファイルに書き出す
        関数
*/
#include <fcntl.h>                    ❶
#include <stdlib.h>
#include <stddef.h>
#define TRANSNUM 16383

#define PSET 0                        ❷
#define PRESET 1
#define COMPLEMENT 2

int gsave0(fh,plane)                  ❸
int fh;
int plane;
{
```

```
unsigned ead;
int number;
int *buffer;                                              ❹
int i;
int dc_reg;      /* DC レジスタ */

/* csrw */
ead=0x4000U+0x4000U*plane;
send_command( 0x49 );                                     ❺
send_word_parameter( ead );

/* MASK */
send_command( 0x4a );                                     ❻
send_word_parameter( 0xffff );

/* vectw */
dc_reg = TRANSNUM;
send_command( 0x4c );                                     ❼
send_parameter( 0x02 );          /* DIR = 2 */
send_word_parameter( dc_reg );

/* read */
send_command( 0xa0 );   /* word transfer , mod=00 */ ❽
buffer = malloc( (size_t) TRANSNUM*2);                    ❾
for(i=0 ;i<=TRANSNUM-1;i++){
        *(buffer+i) =  read_word_data();                  ❿
}

number = write( fh , (char *)buffer , TRANSNUM*2 );       ⓫
free( buffer );                                           ⓬
return( number );
}                                                         ⓭
```

解説

❶必要なヘッダファイルを include します。

❷定数を define します。TRANSNUM に転送ワード数を設定しています。

❸ gsave0( )を定義します。ファイルに書き込んだバイト数を返すため int 型で宣言しています。引数は，ファイルハンドルと VRAM のプレーン番号です。

❹ gsave0( )関数内で使用するローカル変数の宣言です。

❺ CSRW コマンドを送出し，読み出しを開始するアドレスを GDC に指示します。ここでは，グラフィックス VRAM の先頭から読み出すため 4000H，8000H，C000H のいずれか(プレーン番号による)をパラメータ送出します。CSRW コマンドは 49H です。

❻グラフィックスモードでの VRAM データの読み出しなので，MASK レジスタの値を FFFFH にします。MASK コマンドは 4AH です。

❼読み出し方向/読み出しワード数を指定するために，VECTW コマンドを送出します。読み出し方向 DIR=2，読み出しワード数=16383 を設定しています。VECTW コマンドは 4CH です。

❽ READ コマンド(A0H)を送出し，GDC に VRAM データの読み出しを開始させます。これ以後，GDC は FIFO に 16383 ワードのデータを次々に送ってきます。

❾次に送られてきたデータを一時格納するための領域を確保します。転送されてくるデータの量は，TRANSNUM×2 バイトなので，その大きさのメモリを malloc( )関数で確保します。その領域へのポインタは buffer にセットします。

❿ FIFO からデータを read_ word_ data( )関数を使って読み出し，バッファに格納します。

⓫全データが格納し終わったら，write( )関数を使ってファイルに書き出します。実際に書き込まれたバイト数を，変数 number に入れています。

⓬ malloc で確保した領域を free( )関数を使って開放します。

⓭書き込まれたバイト数を戻り値とします。

この gsave0( )関数を使えば，目的の gsaver( )関数は次のように定義できます。

●リスト52

```
/*
        グラフィックスVRAMをセーブする
        関数
*/
void gsave(fname)
char *fname;    /* セーブするファイル名 */          ❶
{
        int fh;
        int number;                                   ❷
        fh = open( fname , O_CREAT | O_RDWR | O_RAW , S_IWRITE |
                                              S_IREAD) ; ❸
        number=gsave0(fh,0);                          ❹
        if( number != TRANSNUM*2 ){
                printf("FILE WRITE ERROR.");          ❺
                exit(1);
        }
        number=gsave0(fh,1);
        if( number != TRANSNUM*2 ){
                printf("FILE WRITE ERROR.");
                exit(1);
        }                                             ❻
        number=gsave0(fh,2);
        if( number != TRANSNUM*2 ){
                printf("FILE WRITE ERROR.");
                exit(1);
        }
        close( fh );                                  ❼
}
```

解説

❶gsave()関数を宣言します。引数はファイル名へのポインタです。
❷ローカル変数を定義します。
❸ファイルを読み書き可能，バイナリーモード(O_RAW)でオープンします。O_RAWでオープンしないと，1AHというデータをエンドオブファイルとみなし，書き込みを終了してしまうなど弊害があります。
❹VRAMの0プレーンのデータをファイルに書き出します。
❺もし，書き出したバイト数がTRANSNUM＊2と違っていた場合は，書き込みエラーなのでそれを知らせてプログラムを終了します。
❻第1プレーン，第2プレーンに対しても同様の処理をします。
❼close()関数でファイルをクローズします。

gsave関数を使ったmain()関数の例をリスト4に示します。ファイル名を聞き，そのファイルにVRAMのデータを書き込むものです。

◉リスト53

```
/*
        gsave()関数の使用例
*/
void main()
{
        char fname[128];

        printf("INPUT FILE NAME ? ");
        scanf("%s",fname);
        gsave(fname);
}
```

VRAMデータの読み出しも，CPUに比べるとかなり速度が遅いといえます。速度が重要ならばCPUでVRAMをアクセスするほうがよいでしょう。

14 VRAMへのデータ書き込み

もちろんGDCには，グラフィックモードでVRAMにデータを直接書き込む機能があります。GDCにはμPD7220とμPD7220Aがありますが，μPD7220Aになってから書き込みの機能は強化されました。ここでは，μPD7220による書き込み機能を使っています(もちろんμPD7220Aでも動作します)。μPD7220を搭載しているPC-9801はPC-9801E/Fなどです。最近のPC-9801は,すべてμPD7220Aです。

前節のgsave( )関数でセーブしたデータを，VRAMにロードする関数

```
void gload(fname);
char *fname;     /* セーブするファイル名 */
```

を定義します。VRAMにデータを書き込む場合，使用するコマンドは

- CSRWコマンド
- MASKコマンド
- VECTWコマンド
- WRITEコマンド

です。

1 CSRWコマンド

データの書き込むアドレスをGDCに指示します。ドットアドレスは指定しません。コマンド/パラメータは，次のとおりです。

C	0	1	0	0	1	0	0	1
P1	←— EAD_L —→							
P2	←— EAD_M —→							
P3	←— dAD —→				WG	0	EAD_H	

2 MASK コマンド

VRAM に転送したいデータを指定します。MASK のコマンド/パラメータは，次のとおりです。

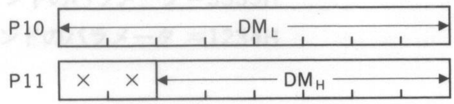
C	0	1	0	0	1	0	1	0
P1	← $MASK_L$ →							
P2	← $MASK_H$ →							

データを毎回 MASK コマンドで GDC に知らせるので，読み出しに比べて，はるかに時間がかかります。

3 VECTW コマンド

GDC に，データの書き込みであることを指示します。コマンド/パラメータは，次ページ上のようになります。

C	0	1	0	0	1	1	0	0
P1	SL	R	C	T	L	← DIR →		
P2	← DC_L →							
P3	×	DGD	← DC_H →					
P4	← D_L →							
P5	×	×	← D_H →					
P6	← $D2_L$ →							
P7	×	×	← $D2_H$ →					
P8	← $D1_L$ →							
P9	×	×	← $D1_H$ →					



このパラメータには，次の値を設定します。

■ DIR

書き込み後に，EADの移動する方向を指示します。ここでは，一応，2を設定していますが，データの書き込みごとに*EAD*を設定しているのであまり意味はありません(図3-44)。

■ DCレジスタ

0を設定してください。DCレジスタには，通常，書き込むデータ数−1(単位はワード)を指定します。もし，DC=9とすれば，1回WRITEするごとに10ワードのデータが書き込まれます。ここでは，データを変えながら1ワードの書き込みを繰り返しているので，DC=0に設定する必要があります。

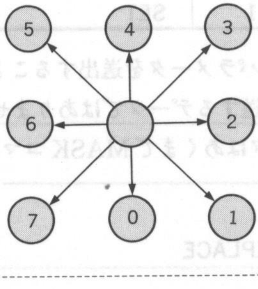
■ Dレジスタ

設定禁止

■ D2レジスタ

設定禁止

●図3-44●



■ D1 レジスタ

設定禁止

■ DM レジスタ

設定禁止

4 WRITE コマンド

WRITEコマンドは，データを書き込むことを指示します。ほかの描画では，ドット修正モードのみを指定するコマンドでしたが，データの書き込みを指示する機能ももっています。ただし，μPD7220の場合は，あまり強力ではありません。WRITEのコマンドは，次のとおりです。

C	0	0	1	WLH	0	MOD

WLH	機能
00	ワード転送
01	設定不可
10	下位バイト転送
11	上位バイト転送

MOD	機能
00	REPLACE
01	COMPLEMENT
10	CLEAR
11	SET

WRITEコマンドに続いてパラメータを送出することで，データを書き込むわけですが，このパラメータは転送するデータではありません。間違えやすいので注意してください。転送するデータはあくまでMASKコマンドで指定します。

たとえば

ドット修正モード =REPLACE

WRITE コマンドのパラメータ＝3333H

MASK コマンドのパラメータ ＝1234H

の場合，3333H のビット 0 は 1 なので，1234H が VRAM に書き込まれます。次にビット 1 は 0 なので，0000H が書き込まれ，これが繰り返されます。つまり，MASK のパラメータの値が，WRITE のパラメータのビットパターンに応じて書き込まれたり，書き込まれなかったりするのです。

VRAM1 プレーンに対して，ファイルのデータを書き込む関数 gload0( ) を定義します。関数 gload0( ) は，リスト 54 のようになります。

◉リスト 54

```
/*
        VRAM1プレーンにデータを書込む
        関数
*/
#include <fcntl.h>
#include <stdlib.h>
#include <stddef.h>                         ❶
#define TRANSNUM 16383

#define PSET 0
#define PRESET 1
#define COMPLEMENT 2

int gload0(fh,plane)
int fh;                                     ❷
int plane;
{
        unsigned ead;
        int number;
```

```
    int *buffer;                                              ❸
    int i;

    ead=0x4000U+0x4000U*plane;                                ❹
    buffer = malloc( (size_t) TRANSNUM*2);                    ❺
    number = read( fh , (char *)buffer , TRANSNUM*2 );        ❻
    for(i=0 ;i<=TRANSNUM-1;i++){                              ❼
            /* csrw */
            send_command( 0x49 );
            send_word_parameter( ead++ );                     ❽
            send_parameter( 0x00 );
            /* MASK */
            send_command( 0x4a );                             ❾
            send_word_parameter( *(buffer+i) );
            /* vectw */
            send_command( 0x4c );
            send_parameter( 0x02 );        /* DIR = 2 */      ❿
            send_parameter( 0x00 );
            send_parameter( 0x00 );
            /* write */
            send_command( 0x20 );   /* word transfer ,
                                        mod=REPLACE * /       ⓫
            send_word_parameter( 0x01 );
    }
    free( buffer );
    return( number );                                         ⓬
}
```

解説

❶ヘッダファイルをincludeします。TRANSNUMに転送ワード数を設定しています。

❷関数gload0()の宣言です。仮引数には，ファイルハンドルと転送するVRAMのプレーン番号がきます。

❸関数gload0()内で使用するローカル変数を宣言します。

❹変数eadに転送を開始するGDCアドレスを計算します。

❺malloc()関数を使ってバッファ領域を動的に確保します。

❻ファイルハンドルの示すファイルからバッファに，TRANSNUM×2バイト転送します。実際に転送されたバイト数は，変数numberに代入しておきます。

❼バッファに読み出したデータをGDCに転送します。

❽まず，CSRWコマンド(49H)を送出します。パラメータは，描画アドレスEADです。変数eadの値をインクリメントし，次の描画に備えます。

❾MASKコマンド(4AH)を送出します。パラメータには，VRAMに転送するデータを与えます。

❿VECTWコマンド(4CH)の送出です。描画方向はDIR＝2にしています。また，転送ワード数は1を設定しています。

⓫WRITEコマンド(20H)を送出し，データをVRAMに書かせます。ワード転送，REPLACEモードを設定しています。パラメータは最下位ビットが1になるように01Hを送っています。

⓬転送が終了したならば，使用したバッファをfree()関数で開放し，実際に転送したバイト数を返して終わります。

関数gload0()を使えば，目的の関数gload()はリスト55のように定義できます。

●リスト55

```
/*
        データをVRAMにロードする関数
 */
void gload(fname)                                              ❶
char *fname;     /* ロードするファイル名 */
{
        int fh;                                                ❷
        int number;
                                                   | S_IREAD) ; ❸
        fh = open( fname , O_CREAT | O_RDWR | O_RAW , S_IWRITE
        number=gload0(fh,0);
        if( number != TRANSNUM*2 ){
                printf("FILE READ ERROR.");                    ❹
                exit(1);
        }
        number=gload0(fh,1);
        if( number != TRANSNUM*2 ){
                printf("FILE READ ERROR.");
                exit(1);
        }                                                      ❺
        number=gload0(fh,2);
        if( number != TRANSNUM*2 ){
                printf("FILE READ ERROR.");
                exit(1);
        }
        close( fh );                                           ❻
}
```

解説

❶関数 gload( )を宣言します。値を返さないので void 型にしています。仮引数はファイル名へのポインタです。

❷ gload( )関数内で使用するローカル変数です。

❸ファイルをバイナリモードでオープンします。

❹プレーン 0 にデータを転送をします。もし，転送バイト数が少なすぎる場合には，"FILE READ ERROR." を表示し，プログラムを終了します。

❺同様にプレーン 1，プレーン 2 へ転送します。

❻転送がうまく行えたならば，close( )関数を使ってファイルをクローズします。

gload( )関数の使用例をリスト 56 に示します。

◉リスト 56

```
*/
void main()
{
        char fname[128];

        start();
        all();
        cls2();
        printf("INPUT FILE NAME ? ");
        scanf("%s",fname);
        gload(fname);
}
```

15 画面フィル

WRITEコマンドを使うと，画面全体をFFFFHで埋めることが比較的簡単に行えます。

VECTWコマンドのパラメータで転送ワード数を設定し，MASKコマンドのパラメータでFFFFHを指定してWRITEコマンドを実行すれば，一定の1次元的領域がFFFFHで埋められます。フローチャートで示せば図3-45のようになります。

MASKレジスタの値を0000Hにすれば，画面消去に使えるというのは間違いです。0000Hの場合，GDCは何もしません。グラフィックモードでは，WRITEコマンドは非力なものです。文字モード/文字グラフィック混在モードでは画面消去に使えます。

これを使ってPC−9801のグラフィック画面をFFFFHで埋める関数fill()を定義します。fill()はリスト57のようになります。

◉リスト57

```
#define TRANSNUM 16383 ――――――――――――――――――――――――― ❶

void fill0(plane) ―――――――――――――――――――――――――――┐
int plane;        ―――――――――――――――――――――――――――┘ ❷
{
        unsigned ead; ――――――――――――――――――――――――――― ❸

        ead=0x4000U+0x4000U*plane; ――――――――――――――――― ❹

        /* csrw */
        send_command( 0x49 );       ――――――――――――――――┐
        send_word_parameter( ead );                   │ ❺
        send_parameter( 0x00 ); ――――――――――――――――――――┘
```

```
    /* MASK */
    send_command( 0x4a );
    send_word_parameter( 0xffff );                                    ❻

    /* vectw */
    send_command( 0x4c );
    send_parameter( 0x02 );        /* DIR = 2 */                      ❼
    send_word_parameter( TRANSNUM-1 );

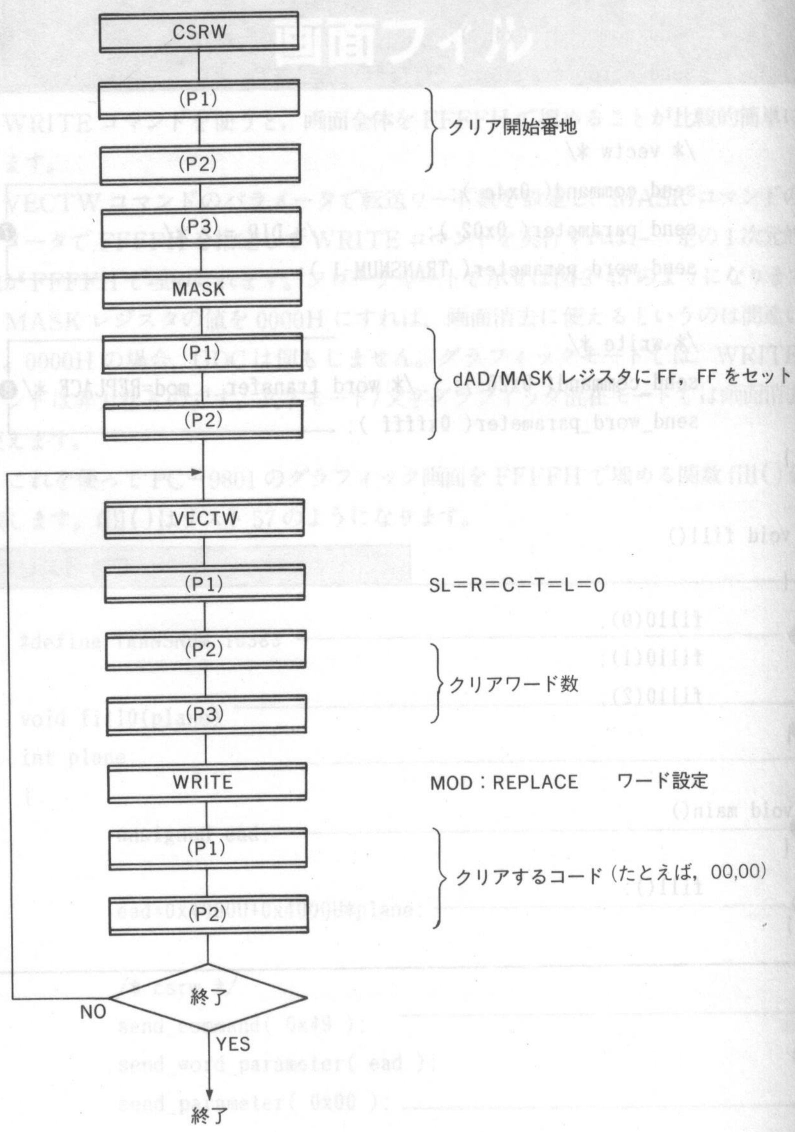
    /* write */
    send_command( 0x20 );   /* word transfer , mod=REPLACE */          ❽
    send_word_parameter( 0xffff );

}

void fill()
{
    fill0(0);
    fill0(1);
    fill0(2);
}

void main()
{
    fill();
}
```

●図 3-45●


CSRW
(P1)
(P2)
クリア開始番地
(P3)
MASK
(P1)
(P2)
dAD/MASK レジスタに FF，FF をセット
VECTW
(P1)
SL=R=C=T=L=0
(P2)
(P3)
クリアワード数
WRITE
MOD：REPLACE　ワード設定
(P1)
(P2)
クリアするコード（たとえば，00,00）
終了
NO
YES
終了

解説

❶転送ワード数をTRANSNUMとします。

❷VRAM1プレーンに対してfillを行う関数fill0( )を定義します。引数はプレーン番号です。

❸ローカル変数を宣言します。

❹fillを開始するアドレスを変数eadに代入します。

❺CSRWコマンドでそのアドレスをGDCに知らせます。

❻MASKコマンドでfillするパターンを送出します。

❼VECTWコマンドで描画方向，fillするワード数を指定します。このように実際の転送ワード数−1を指定します。

❽WRITEコマンドを送出しfillを開始させます。ワード転送，REPLACEモードを指定しています。

fill0( )関数を3回呼ぶことで，fill( )関数は定義できます(リスト58)。

◉リスト58

```
/*
  画面をFFFFHで埋める関数
*/
void fill()
{
        fill0(0);
        fill0(1);
        fill0(2);
}
```

fill( )を呼ぶプログラムは，たとえばリスト59のようになります。

◉リスト 59

```
/*
        関数fill()のテスト
*/
void main()
{
        fill();
}
```

このプログラムを実行するとわかるように，fillはかなり高速です。

第4章
高速化

PC-9801 で GDC を使ったプログラムを書く場合には，高速化するノウハウがいくつかあります。

- EGC を使う
- フラッシュ描画
- GDC5MHz モード
- アセンブラで書く
- そのほか

これらを組み合わせて使うのがよいでしょう。

1 EGCを使う

PC-9801VX以降の機種には，EGC(Enhanced Graphic Charger)が搭載されています。PC-9801UV/VF/VMにはGRCG(Graphic CharGer)というものが搭載されていましたが，このグラフィックチャージャを改良したものがEGCというわけです。さて，このEGCを使うとなにができるのでしょうか。EGCの機能は，次のとおりです。

①GRCGとの互換モード

②4面のVRAMの同時制御機能

③ラスターオペレーション機能

④VRAM内のデータ移動のビット単位シフト機能

①の「GRCG互換モード」が高速化に大きな力を発揮してくれます。グラフィックチャージャは，VRAMの同時書き込み機能をもっていました。しかし，それはCPUからしか利用できませんでした。EGCは，GRCGとの互換モードをもち，さらにそれをGDCから使用することができるようになっています。EGCを使えば，単純計算で4倍まで高速にすることができます。これまで描画を行うときはR，G，Bの各VRAMプレーンに1回ずつ，計3回描画していました。リスト1は，描画例で示したラインルーチンの一部ですが，色に応じてR，G，Bの各プレーンに描画しているのがわかります。

◉リスト 1

```
/*
          EGCを使用しないライン関数(部分)
*/
void line(x1,y1,x2,y2,col)
unsigned x1,y1,x2,y2,col;
{
          if( (col & 1) != 0 )     /* BLUE */
                    line0(x1,y1,x2,y2,0,PSET);
          else
                    line0(x1,y1,x2,y2,0,PRESET);
          if( (col & 2) != 0 )     /* RED */
                    line0(x1,y1,x2,y2,1,PSET);
```

```
        else
                lineO(x1,y1,x2,y2,1,PRESET);
        if( (col & 4) != 0 )        /* GREEN */
                lineO(x1,y1,x2,y2,2,PSET);
        else
                lineO(x1,y1,x2,y2,2,PRESET);
}
```

ほかの描画関数もすべてこんな感じになっています。これは，すべてのPC-9801で動作させるためで，EGCを搭載したPC-9801ならば，これが1回で行えるわけです。RGBといいましたが，PC-9801はRGBのほかに，もう1プレーンもっています。 正確には「4画面同時書き込み機能をもつ」といえます。この同時書き込み機能を使うときには注意しなければならないことがあります。それはタイルレジスタを変更するタイミングです。GDCが描画中に変更すると途中で色が変わってしまいます。だから，GDCの描画終了を待ってから変えなければなりません。しかし，それではCPUが無駄に待つことになるので，高速なプログラムを作成するには，同じ色の描画をまとめて行うのがよいでしょう。

表4-1に，グラフィックチャージャのI/Oアドレスとデータの意味を示します。

GDCから4画面同時書き込みを使用する場合は

CGモード	1
RMWモード	1
P3EN	0
P2EN	0
P1EN	0
P0EN	0

なので，ライトモードレジスタには

11000000B ＝ C0H

をOUTします。また，タイルレジスタに描画したいプレーンにはFFHを，消去したいプレーンには00Hを設定します。変数colorの値によってグラフィックチャージャを設定するプログラムはBASIC的に書くと，

●表 4-1

命　令	I/O アドレス	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
ライトモードレジスタ	7C	CG モード	RMW モード	0	0	$\overline{\text{P3EN}}$	$\overline{\text{P2EN}}$	$\overline{\text{P1EN}}$	$\overline{\text{P0EN}}$
ライトタイルレジスタ	7E	タイルレジスタ 0 〜 3*							

＊　モードレジスタにライトを行うと，タイルレジスタ 0 ライトにリセットされる
　その後ライトするたびに，書かれるレジスタはレジスタ 1，レジスタ 2，レジスタ 3，レジスタ 0，と変化する

ビット名	ビット＝1 の意味	ビット＝0 の意味
CG モード	GRCG を有効とする CPU の VRAM アクセスをきっかけとして，GRCG が各モードの動作を実行する	GRCG を無効とする CPU の VRAM アクセスは，そのまま VRAM のリード(ライト)となる
RMW モード	CPU の VRAM ライトにより，RMW モードの動作を行う CPU の VRAM リードは無視される	CPU の VRAM ライトにより，TDW モード　の動作を行う CPU の VRAM リードにより TCR モードの動作を行う
P3EN, P2EN P1EN, P0EN	該当するプレーンを無効とする	該当するプレーンを有効とする 複数ビットの指定が可能 GRCG は，有効となっているプレーンに対してのみアクセスを行う

```
1000 OUT &H7C,&HC0
1010 IF (COLOR AND 1)=0 THEN OUT &H7E,0 ELSE OUT &H7E,&HFF
1020 IF (COLOR AND 2)=0 THEN OUT &H7E,0 ELSE OUT &H7E,&HFF
1030 IF (COLOR AND 4)=0 THEN OUT &H7E,0 ELSE OUT &H7E,&HFF
1040 IF (COLOR AND 8)=0 THEN OUT &H7E,0 ELSE OUT &H7E,&HFF
```

のようになります。このようにタイルレジスタは，7CH にデータが OUT されるとレジスタ 0 ライトにリセットされ，以後 7EH にデータが書き込まれるごとにレジスタ 1，レジスタ 2，レジスタ 3 レジスタ 0 と変わっていきます。

リスト 2 に EGC を利用した直線描画関数 line_ egc( )を示します。

◉リスト２

```
/*
  EGCを利用した直線描画関数
*/

#include <dos.h>
#define PSET 0
#define PRESET -1

void line(x1,y1,x2,y2,col)
int x1,y1,x2,y2,col;
{
        unsigned ead;
        unsigned dad;
        int dir;
        int temp;
        int dx,dy;
        int dc,d2,d1,d;

        /* textw */
        send_command( 0x78 );   /* textw command */
        send_word_parameter( 0xffff );  /* pattern */

        /* write */
        send_command( 0x23 );

        /* init */
        if( x1 > x2 ){
                temp=x2; x2=x1; x1=temp;
                temp=y2; y2=y1; y1=temp;
        }
        if( x1 == x2 )
                if( y1 > y2 ){
                        temp=y2 ; y2=y1 ; y1=temp;
                };


        /* culadr */

        ead = 0X4000 + 40*y1 + x1/16;
        dad = x1 % 16;

        /* csrw */
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( 16*dad );
```

```
        /* caldir */
        dx = x2 - x1;
        dy = y2 - y1;
        if( dy > 0 ){
                if( dy > dx ){
                        dir = 0;
                        temp = dy ; dy = dx ; dx = temp;
                }
                else
                        dir = 1;
        }
        else{
                dy = -dy;
                if( dy < dx )
                        dir = 2;
                else{
                        dir = 3;
                        temp = dy; dy = dx; dx=temp;
                }
        }

        /* culpar */
        dc = dx;
        d = 2*dy - dx;
        d2 = 2*dy-2*dx;
        d1 = 2*dy;

        /* vectw */
        send_command( 0x4c );
        send_parameter( 8 + dir );
        send_word_parameter( dc );
        send_word_parameter( d );
        send_word_parameter( d2 );
        send_word_parameter( d1 );

/*
        EGCの設定
  */
        while( (inp(0xa0) & 8) != 0);    /* DRAWING */      ❶
        outp( 0x7c , 0xc0 );                                ❷
        if( (col & 1) != 0)                                 ❸
                outp( 0x7e , 0xff );
        else
                outp( 0x7e , 0x00 );
        if( (col & 2) != 0)
                outp( 0x7e , 0xff );
```

```
        else
                outp( 0x7e , 0x00 );
        if( (col & 4) != 0)
                outp( 0x7e , 0xff );
        else
                outp( 0x7e , 0x00 );
        if( (col & 8) != 0)
                outp( 0x7e , 0xff );
        else
                outp( 0x7e , 0x00 );

        /* vecte */
        send_command( 0x6c );
}
/*
        line()関数のテスト
*/
void main()
{
        int i;
        start();
        all();
        cls2();
        for(i=0;i<=639;i++)
                line(i,0,639-i,399,i % 8);
        for(i=0;i<=399;i++)
                line(639,i,0,399-i,i % 8);

        while( (inp(0xa0) & 8) != 0);   /* DRAWING */
        outp( 0x7c , 0x00 );    /* GRCG OFF */
}
```

❹

解説

❶GDCが描画中でないことを確認します。DRAWINGはステータスフラグのビット3です。

❷ライトモードレジスタ(7CH)にCGモード，RMWモード，全VRAMへの描画有効を指示します。

❸プレーン0に対して描画する場合はライトタイルレジスタ(7EH)に，FFHを消去する場合は00Hを送出します。

❹同様にプレーン1，プレーン2，プレーン3に対しても同じ処理をします。

グラフィックチャージャをセットしている位置に注意してください。プログラムではVECTEコマンドの直前で設定しています。もし，line( )関数の先頭で設定しようとすると，GDCが直前の描画をまだ終了していないため，描画終了まで待つ時間がかかります。その無駄時間を避けるために，VECTEコマンドの直前で設定しているわけです。もし，同じ色だとわかっているならば，グラフィックチャージャを設定しなおす必要はありません。

これでグラフィックチャージャの設定は終わりです。あとはGDCで描画を実行します。グラフィックチャージャが効いているので，描画はどのプレーンに対して行っても同じ結果になります。このプログラムでは，プレーン0に対して描画を行っています。描画開始アドレス/描画開始ドットアドレスの計算部が描画例のものと違い，このように簡単になります。

VECTEコマンドが一度しか送出されてないことからもわかるように，描画は一度しか実行していません。

なお，main( )関数の終わりでグラフィックチャージャの機能を停止させています。

描画例で紹介したEGCを利用しないline( )関数との速度比をリスト3のプログラムで調べてみました。

◉リスト3

```
/*
        ベンチマーク
*/
void main()
{
        int i;
        int j;
        start();
        all();
        cls2();


        for(j=1;j<=10;j++){
                for(i=0;i<=639;i++)
                        line(i,0,639-i,399,i % 8);
                for(i=0;i<=399;i++)
                        line(639,i,0,399-i,i % 8);
        }
}
```

プログラム実行にかかった時間は，次のようになりました。

①GDCのみ ……………………………35秒
②グラフィックチャージャ使用 ……… 13秒

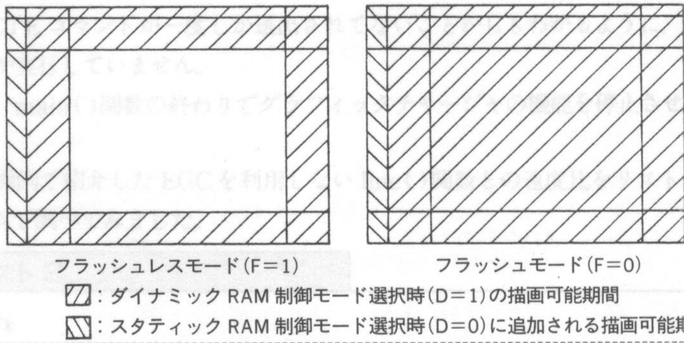
(PC-9801LS　CPU80386　16MHzの場合)

①と②を比較するとわかるように，グラフィックチャージャを使用すると約3倍高速化されているのがわかります。

2 フラッシュ描画を使う

GDCには，フラッシュ描画とフラッシュレス描画があります。フラッシュ描画とは，あらゆるときに描画できるモードで，走査線が表示中でも描画できます。そのため表示ロジックとのデータの競合が起こり，画面がちらつきます。フラッシュレス描画はこのちらつきをなくしたモードで，表示期間中にGDCは描画をしません(図4-1)。

●図4-1●



つまり，描画モードでは

描画モード	速度	画面
フラッシュ描画	高速	ちらつきあり
フラッシュレス描画	低速	ちらつきなし

という関係があります。フラッシュ描画とフラッシュレス描画の速度比は5倍です。使う側からいえば，ちらつきがなくてフラッシュ描画と同じ速度が得られれば一番よいわけです。

最近のPC-9801は，デュアルポートRAMをVRAMに搭載しています。このデュアルポートRAMのおかげで，PC-9801はGDCでフラッシュ描画を行っても画面がちらついたりはしません。だから，遠慮なくフラッシュ描画を指示するべき

です。フラッシュ描画の指定は，SYNCのパラメータで行います。

C	0	0	0	0	1	1	1	DE
P1	0	0	CHR	F	1	D	G	S
P2	← C/R →							
P3	← VS_L →			← HS →				
P4	← HFP →						← VS_H →	
P5	DS	PH	← HBP →					
P6	VH	VL	← VFP →					
P7	← L/F_L →							
P8	← VBP →						← L/F_H →	

フラッシュ描画を行う関数 flash( )，フラッシュレス描画を行う flashless( )は，次のようになります。

◉リスト 4

```
/*
        フラッシュ描画，フラッシュレス描画
      を指示する関数
*/
void flash()
{
        send_command( 0x0f );   /* SYNC COMMAND   */
                                /* DISPLAY ENABLE */
        send_parameter( 0x16 ); /* CHR F I D G S  */
                                /*  0  1 0 1 1 0  */
}

void flashless()
{
        send_command( 0x0f );   /* SYNC COMMAND   */
                                /* DISPLAY ENABLE */
        send_parameter( 0x06 ); /* CHR F I D G S  */
                                /*  0  1 0 1 1 0  */
}
```

デュアルポートRAMを搭載したPC-9801は，電源投入時にフラッシュモードに設定されています。そのため，SYNCのパラメータを操作していないときは高速モードになっています。

直線を10400本描画するベンチマークテストをしたところ

フラッシュモード	6秒
フラッシュレスモード	20秒

(PC-9801LS　CPU80386　16MHz　グラフィックチャージャ使用時)

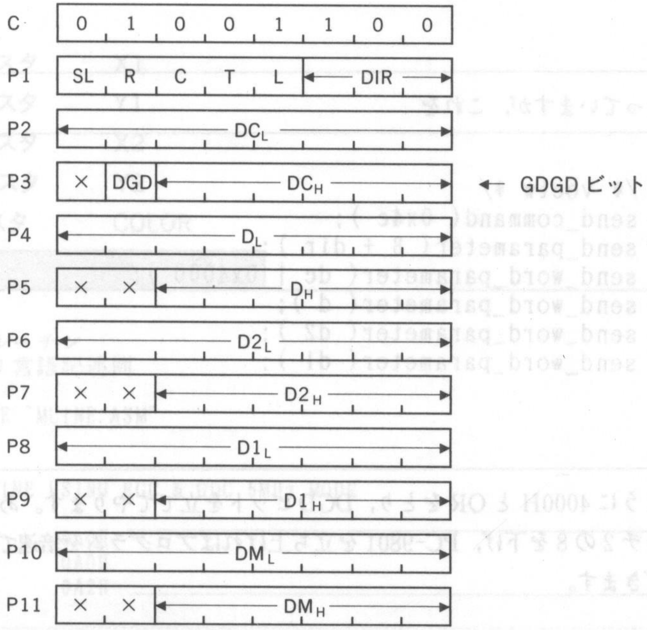
という結果がでました。実際のプログラムでも，3倍以上高速化されるわけです。

3

GDC5MHzモードを使用する

PC-9801のGDCは，通常2.5MHzのクロックで動作しています。これを倍の5MHzで動作させることができます(PC-9801VX以降)。クロック周波数を5MHzにすることで描画速度も倍になります。倍速にするもっとも簡単な方法は，ディップスイッチで設定する方法です。ディップスイッチ2のビット8をON(下に下げる)にして，PC-9801をリセットします。すると，PC-9801は，GDCを5MHzに初期化して立ち上がります。

GDC5MHzの場合，描画する際に，ひとつだけ注意すべき点があります。VECTWコマンドのパラメータにDGD(Dynamic Graphics Drawing)ビットというのがありました。



前ページに示したパラメータ3のビット6がDGDビットです。GDC5MHzモードのときは，このDGDビットを必ず1にしてください。DGDビットが1のとき，水平方向以外に描画アドレスが進行する場合は，±P/2のオフセットアドレスが加算されます。通常は，±Pが加算されていましたが，GDC5MHzの場合，初期化時にPITCHの値を通常の倍の値に設定しています。そのため正常に描画させるには，DGD=1として±P/2ずつ変化させる必要があるわけです。

描画例のプログラムで変更する箇所は，ひとつだけです。VECTWコマンドの送出部は

```
/* vectw */
send_command( 0x4c );
send_parameter( 8 + dir );
send_word_parameter( dc );
send_word_parameter( d );
send_word_parameter( d2 );
send_word_parameter( d1 );
```

となっていますが，これを

```
/* vectw */
send_command( 0x4c );
send_parameter( 8 + dir );
send_word_parameter( dc | 0x4000 );
send_word_parameter( d );
send_word_parameter( d2 );
send_word_parameter( d1 );
```

のように4000HとORをとり，DGDビットを立ててやります。あとはディップスイッチ2の8を下げ，PC-9801を立ち上げればプログラムを倍速で動作させることができます。

4 アセンブリ言語で書く

これまでは，C言語でプログラムを作成してきました。プログラムの組みやすさ，わかりやすさからいえば，高級言語のほうがよいわけですが，実行速度の面から見ればアセンブリ言語で記述するのが一番です。パラメータ計算などが高速化されれば，GDCのFIFOが常にいっぱいになり，GDCに休む暇を与えないことになります。

リスト5に，アセンブリ言語でラインルーチンMLINEを記述した例を示します。なお，GDC5MHzモード用でEGCのグラフィックチャージャ互換機能を使用しています。

MLINEルーチンの使い方は，次のようになっています。

MLINE：NEAR

入力

AXレジスタ	X1
BXレジスタ	Y1
CXレジスタ	X2
DXレジスタ	Y2
SIレジスタ	COLOR

◉リスト5

```
;
; 直線描画ルーチン
; アセンブリ言語記述例
;
;       SAVE "MLINE.ASM"
;
;
; LINE ROUTINE USING EGC & GDC 5MHz MODE
;
STATPT  EQU     0A0H
POUTPT  EQU     0A0H                              ❶
COUTPT  EQU     0A2H
```

```
CODE    SEGMENT BYTE PUBLIC                                              ❷
        ASSUME  CS:CODE,DS:CODE
        PUBLIC  MLINE
MLINE:
        MOV     X1,AX
        MOV     Y1,BX                                                    ❸
        MOV     X2,CX
        MOV     Y2,DX
        MOV     COLOR,SI

TEXTW:  MOV     AL,78H
        MOV     COMMAND,AL                                               ❹
        CALL    OUTC
        MOV     WRDPAR,0FFFFH
        CALL    OUTWP

WRITE:
        MOV     AL,20H          ; 20H REPLACE COMMAND                    ❺
        MOV     COMMAND,AL
        CALL    OUTC

INIT:                           ; LET DELTA_X=X2-X1>=0
        MOV     AX,X2                                                    ❻
        CMP     AX,X1
        JGE     INIT1           ; IF DELTA_X>=0 THEN INIT1

        MOV     BX,X1           ; SWAP X1,X2
        MOV     X1,AX
        MOV     X2,BX
        MOV     AX,Y1           ; SWAP Y1,Y2                             ❼
        MOV     BX,Y2
        MOV     Y2,AX
        MOV     Y1,BX
INIT1:  MOV     AX,X1
        CMP     AX,X2                                                    ❽
        JNE     INIT2           ; IF X1<>X2 THEN INIT2

        MOV     AX,Y2
        CMP     AX,Y1                                                    ❾
        JGE     INIT3           ; IF Y2>=Y1 THEN INIT3

        MOV     BX,Y1           ; SWAP Y1,Y2
        MOV     Y1,AX                                                    ❿
        MOV     Y2,BX
INIT2:
INIT3:

CULADR:                                                                  ⓫
        MOV     CX,4000H        ; LET V-RAM START ADRESS=4000H
```

```
        MOV     AX,X1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1                 ; AX=INT(X1/16)            ⑫

        ADD     CX,AX                                            ⑬

        MOV     AX,Y1
        MOV     BX,AX
        SHL     BX,1
        SHL     BX,1
        ADD     AX,BX                                            ⑭
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1                 ; AX=40*Y1

        ADD     CX,AX                                            ⑮
        MOV     EAD,CX

        MOV     AX,X1
        AND     AX,0FH               ; AX=X1 MOD 16              ⑯
        MOV     DAD,AL
;
CSRW:
        MOV     AL,49H               ; CSRW COMMAND=49H
        MOV     COMMAND,AL
        CALL    OUTC

        MOV     AX,EAD
        MOV     PARMTR,AL            ; COMMAND OUT EADL
        CALL    OUTP                                             ⑰
        MOV     PARMTR,AH            ; COMMAND OUT EADH
        CALL    OUTP

        MOV     AL,DAD
        MOV     CL,4
        SHL     AL,CL                ; SHIFT DAD 4 TIMES
        MOV     PARMTR,AL
        CALL    OUTP
;
CULDIR:
        MOV     AX,X2
        SUB     AX,X1
        MOV     DELTA_X,AX                     ; DELTA_X=X2-X1   ⑱

        MOV     AX,Y2
        SUB     AX,Y1
        MOV     DELTA_Y,AX                     ; DELTA_Y=Y2-Y1
```

```
        MOV     AX,DELTA_Y                                       ⑲
        CMP     AX,0
        JLE     DIR1            ; IF DELTA_Y<=0 THEN DIR1

        MOV     BX,DELTA_X                                       ⑳
        CMP     AX,BX
        JLE     DIR2

        MOV     DIR,0           ; DIR=0                          ㉑

        MOV     AX,DELTA_X
        MOV     BX,DELTA_Y
        MOV     DELTA_Y,AX
        MOV     DELTA_X,BX                ; SWAP DELTA_X,DELTA_Y

        JMP     DIR3

DIR2:   MOV     DIR,1                                            ㉒
        JMP     DIR3

DIR1:   NEG     DELTA_Y                                          ㉓
        MOV     BX,DELTA_Y

        MOV     AX,DELTA_X
        CMP     BX,AX
        JG      DIR4            ; IF DELTA_Y>DELTA_X THEN DIR4

        MOV     AL,2                                             ㉔
        MOV     DIR,AL          ; DIR=2
        JMP     DIR3

DIR4:   MOV     DIR,3                                            ㉕

        MOV     AX,DELTA_X
        MOV     BX,DELTA_Y
        MOV     DELTA_Y,AX
        MOV     DELTA_X,BX      ; SWAP DELTA_X,DELTA_Y

DIR3:
;
CULPAR:                                                          ㉖
        MOV     AX,DELTA_X
        MOV     DC_REG,AX       ; LET DC_REG=DELTA_X

        MOV     AX,DELTA_Y                                       ㉗
        SAL     AX,1
        SUB     AX,DELTA_X
        MOV     D_REG,AX        ; LET D_REG=2*DELTA_Y-DELTA_X
```

```
        MOV     AX,DELTA_Y                                          ┐
        SUB     AX,DELTA_X                                          │㉘
        SAL     AX,1                                                │
        MOV     D2_REG,AX          ; LET D2_REG=2*DELTA_Y-2*DELTA_X ┘
        MOV     AX,DELTA_Y                                          ┐
        SAL     AX,1                                                │㉙
        MOV     D1_REG,AX          ; LET D1_REG=2*DELTA_Y           ┘
;
VECTW:
        MOV     AL,4CH             ; VECTW COMMAND=4CH              ┐
        MOV     COMMAND,AL                                          │㉚
        CALL    OUTC                                                ┘
        MOV     AL,DIR                                              ┐
        OR      AL,8               ; DECLEAR TO DRAW 'LINE'         │㉛
        MOV     PARMTR,AL                                           │
        CALL    OUTP                                                ┘
        MOV     AX,DC_REG                                           ┐
        OR      AH,64              ; DGD BIT ON                     │㉜
        MOV     WRDPAR,AX                                           │
        CALL    OUTWP                                               ┘
        MOV     AX,D_REG                                            ┐
        MOV     WRDPAR,AX                                           │㉝
        CALL    OUTWP                                               ┘
        MOV     AX,D2_REG                                           ┐
        MOV     WRDPAR,AX                                           │㉞
        CALL    OUTWP                                               ┘
        MOV     AX,D1_REG                                           ┐
        MOV     WRDPAR,AX                                           │㉟
        CALL    OUTWP                                               ┘
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H                                             ┐
        TEST    AL,8                                                │㊱
        JNZ     GC1                                                 ┘
        MOV     AL,0C0H                                             ┐
        OUT     07CH,AL                                             ┘㊲
        MOV     TILE0,0                                             ┐
        MOV     TILE1,0                                             │㊳
        MOV     TILE2,0                                             │
        MOV     TILE3,0                                             ┘
```

```
        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH                                    ㊴
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3                                            ㊵
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL                                       ㊶
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL
VECTE:
        MOV     AL,6CH              ; VECTE COMMAND=6CH
        MOV     COMMAND,AL                                    ㊷
        CALL    OUTC
;
        RET
;
;       END OF LINE ROUTINE
        ;
;
;       COMMAND OUT ROUTINE FOR GDC
;
OUTC:
;
OUTCLP: IN      AL,STATPT                                     ㊸
        AND     AL,2
        JNZ     OUTCLP              ; WAIT UNTIL FIFO NOT FULL
;
        MOV     AL,COMMAND
        OUT     COUTPT,AL                                     ㊹
;
        RET
;
;       PARAMETER OUT ROUTINE FOR GDC
```

```
;
OUTP:
;
OUTPLP: IN      AL,STATPT                                (45)
        AND     AL,2
        JNZ     OUTPLP          ; WAIT UNTIL FIFO NOT FULL
;
        MOV     AL,PARMTR
        OUT     POUTPT,AL                                (46)
;
        RET
;
;
OUTWP:
        MOV     AX,WRDPAR
        MOV     PARMTR,AL
        CALL    OUTP                                     (47)
        MOV     PARMTR,AH
        CALL    OUTP

        RET

X1      DW      0
Y1      DW      0
X2      DW      0
Y2      DW      0
PARMTR  DB      0
COMMAND DB      0

COLOR   DW      0
EAD     DW      0
DAD     DB      0
DELTA_X DW      0
DELTA_Y DW      0
DIR     DB      0                                        (48)
DC_REG  DW      0
D_REG   DW      0
D1_REG  DW      0
D2_REG  DW      0
WRDPAR  DW      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0

CODE    ENDS
        END
```

解説

❶定数をEQU擬似命令で定義します。

❷セグメント名をCODEとし，ラベルMLINEが外部からアクセスできるようにPUBLIC宣言します。

❸以下，MLINEルーチンの定義です。引数はAX，BX，CX，DX，SIレジスタで渡されるので，まずそれらを変数X_1，Y_1，X_2，Y_2，COLORに保存します。

❹TEXTWのコマンド/パラメータを送出します。コマンドの送出ルーチンはOUTC，パラメータの送出はバイト単位の場合OUTP，ワードのときはOUTWPを使います。これらはあとで定義しています。コマンドには78H，パラメータには0FFFFHを送出しています。

❺WRITEコマンドを送出します。グラフィックチャージャを使用するので，描画色にかかわらず，ドット修正モードはREPLACE(20H)を使用しています。

❻変数X_2とX_1を比較し，$X_2>=X_1$ならばINIT1にジャンプします。

❼$X_2<X_1$の場合はX_1とX_2，Y_1とY_2を交換します。

❽$X_1<>X_2$ならばINIT2に分岐します。

❾$X_1=X_2$の場合，$Y_2>=Y_1$ならばINIT3に分岐します。

❿$X_1=X_2$で$Y_2<Y_1$ならばY_1とY_2を入れ替えます。

⓫描画開始アドレスEADとドットアドレスdADを算出します。グラフィックチャージャを使用するため，描画アドレスはプレーン0に固定しています。CXレジスタにEADを求めることにします。まず，CXに4000Hを代入します。

⓬AX=INT(X1/16)の計算をします。16が2の倍数なのでシフト動作で除算を置き換えています。

⓭これをCXレジスタに加えます。

⓮AXレジスタに40 * Y1を求めます。40=5 * 2 ^ 3なので，乗算をシフトと加算に置き換えています。

⓯これをCXレジスタに加えて変数EADに格納します。

⓰dADを求めます。dAD=X1　%　16なので，AND命令で置き換えています。

⓱いま計算したEAD，dADをCSRWコマンド/パラメータでGDCに送出します。CSRWコマンドは49Hです。

⓲X座標の差をDELTA_X，Y座標の差をDELTA_Yに計算します。

⑲もし，DELTA_Y＜＝0ならばDIR1にジャンプします。

⑳DELTA_Y＜＝DELTA_XならばDIR2にジャンプします。

㉑ DELTA_Y＞＝0 で DELTA_Y＞DELTA_X ならば DELTA_X，DELTA_Y を入れ替え，描画方向DIRを0にしラベルDIR3にジャンプします。

㉒DELTA_Y＜DELTA_XならばDIR＝1とし，DIR3にジャンプします。

㉓DELTA_Y＜0ならばDELTA_Yの符号を反転します。もし，DELTA_Y＞DELTA_XならばDIR4に分岐します。

㉔さもなければ，DIR＝2にしてDIR3にジャンプします。

㉕DELTA_Y＞DELTA_XのときDELTA_X，DELTA_Yを交換し，DIR＝3にします。

㉖DC_REG＝DELTA_Xにします。

㉗D_REG＝2＊DELTA_Y-DELTA_Xにします。

㉘D2_REG＝2＊(DELTA_Y-DELTA_X)です。

㉙D1_REG＝2＊DELTA_Yです。

㉚VECTWコマンド4CHを送出します。

㉛描画方向をパラメータ送出します。

㉜DCレジスタの値を送出します。GDC5MHzモードを使用するため，DGDビットを1にしています。

㉝Dレジスタの値を送出します。

㉞D2レジスタの値を送出します。

㉟D1レジスタの値を送出します。

㊱グラフィックチャージャの設定を行います。まずGDCが描画中でないことを確認します。

㊲グラフィックチャージャをアクティブにします。描画はすべてのプレーンに対して行います。

㊳タイルレジスタの値を入れる変数TILE0，TILE1，TILE2，TILE3を0にします。

㊴変数COLORのビット0のONのとき，TILE0の値を0FFHにします。

㊵同様にTILE1，TILE2，TILE3を設定します。

㊶TILE0～TILE3をグラフィックチャージャに送出します。

㊷グラフィックチャージャの設定ができたので，VECTEコマンド(6CH)を送出し

て描画を開始します。

㊸コマンドを GDC に送出するルーチン OUTC を定義します。まず FIFO が FULL でないことを確認します。

㊹変数 COMMAND の値を OUT します。

㊺パラメータを送出するルーチンです。FIFO が FULL でないことを確認します。

㊻変数 PARMTR の値を OUT します。

㊼ワード長のパラメータを送出するルーチンの定義です。変数 WRDPAR の内容を，上位バイト下位バイトに分けて，OUTP ルーチンを2度呼び出しています。

㊽以下変数の定義です。

MLINE ルーチンを使った例をリスト6に示します。

◉リスト6

```
;
; MLINEルーチンを使用例
;
;        SAVE "DEMO.ASM"
;
; LINE ROUTINE TEST
;
;
; LINE ROUTINE TEST
;
CODE     SEGMENT BYTE PUBLIC        ─┐
         ASSUME  CS:CODE,DS:CODE     │
         EXTRN   MLINE:NEAR          │ ❶
         ORG     100H                │
START:                              ─┘
         MOV     AH,40H   ; DISP START ─┐ ❷
         INT     18H                  ─┘

         MOV     AH,42H   ; 640*400  ─┐
         MOV     CH,0C0H              │ ❸
         INT     18H                 ─┘

         MOV     I,0                 ── ❹
L1:
         MOV     AX,I                ─┐
         MOV     X1,AX                │
         MOV     Y1,0                 │ ❺
```

```
        MOV     AX,639
        SUB     AX,I
        MOV     X2,AX
        MOV     Y2,399
        MOV     AX,I                                    ⑥
        CWD
        MOV     BX,7
        DIV     BX
        INC     DX                                      ⑦
        MOV     COLOR,DX

        MOV     AX,X1
        MOV     BX,Y1
        MOV     CX,X2
        MOV     DX,Y2                                   ⑧
        MOV     SI,COLOR
        CALL    MLINE

        INC     I
        CMP     I,639                                   ⑨
        JLE     L1

        MOV     I,0                                     ⑩
L2:
        MOV     X1,639
        MOV     AX,I
        MOV     Y1,AX
        MOV     X2,0                                    ⑪
        MOV     AX,399
        SUB     AX,I
        MOV     Y2,AX
        MOV     AX,I
        CWD
        MOV     BX,7
        DIV     BX                                      ⑫
        INC     DX
        MOV     COLOR,DX

        MOV     AX,X1
        MOV     BX,Y1
        MOV     CX,X2
        MOV     DX,Y2                                   ⑬
        MOV     SI,COLOR
        CALL    MLINE

        INC     I
        CMP     I,399
        JLE     L2
```

```
        MOV     AL,00H  ; STOP GRAPHIC CHARGER   ⓮
        OUT     07CH,AL
        MOV     AX,04C00H                        ⓯
        INT     21H

I       DW      0                                ⓰
X1      DW      0
Y1      DW      0
X2      DW      0
Y2      DW      0
COLOR   DW      0

code    ends
        END     START
;
; 実行方法
;
;       A:¥>MASM MLINE;
;       A:¥>MASM DEMO;
;       A:¥>LINK DEMO MLINE;
;       A:¥>EXE2BIN DEMO DEMO.COM
;       A:¥>DEMO
;
```

解説

❶セグメント名をMLINEと同じCODEにします。

❷BIOSを使って表示を開始します。

❸同じく，BIOSで640×400モードにします。

❹ループ変数Iを0にします。

❺X_1=I，Y=0，X_2=639-I，Y_2=399に設定します。

❻❼COLOR=I % 7+1を設定します。

❽AX，BX，CX，DX，SIレジスタに値をセットし，MLINEを呼び出します。

❾これを I<=639 の間，繰り返します。

❿ループ変数Iを0にします。

⓫X_1=639，Y_1=I，X_2=0，Y_2=399-Iとします。

⓬COLOR=I % 7+1を設定します。

⑬AX，BX，CX，DX，SIレジスタに値をセットしMLINEを呼び出します。
⑭描画が終了したならば，グラフィックチャージャを停止させます。
⑮MS-DOSに復帰します。
⑯以下，変数の定義です。

アセンブリ言語の長所は，実行速度が高速なことであり，短所は，作りづらく読みにくいことです。またデバッグも大変です。高級言語に比べて，ソースプログラムが冗長になることも否めません。C言語からアセンブリ言語にプログラムを変えた場合，速度の向上は2割程度のようです。GDCのプログラムを高速化したい場合は，まずグラフィックチャージャを使用することを考えるべきでしょう。次に，

●表4-2

単位：JMP$＋2挿入数

周辺LSI名		リカバリータイム (nSEC)	WR→WR 8MHz	WR→WR 10MHz	RD→RD 8MHz	RD→RD 10MHz	WR→RD 8MHz	WR→RD 10MHz	RD→WR 8MHz	RD→WR 10MHz
8237	DMAC	400	1	1	0	0	1	1	0	0
8253	タイマ	1000	1	1	1	1	1	1	1	1
8255A	PIO	850	1	1	1	1	1	1	1	1
8259/71059	PIC	190(500)	1/0	1/0	0	0	1	1	0	0
8251A SIO	モード初期化	6tcy*1	3	3	0	0	0	0	0	0
8251A SIO	ライトデータ非同期	8tcy*1	4	4	0	0	0	0	0	0
8251A SIO	ライトデータ同期	16tcy*1	7	7	0	0	0	0	0	0
765AC	FDC		0	0	0	0	0	0	0	0
7220A	グラフ 2.5Mモード	標準CRT 1118	2	3	2	2	2	3	2	2
7220A	グラフ 2.5Mモード	高解像CRT 760	2	2	1	1	2	2	1	1
7220A	テキスト/グラフ 5Mモード	標準CRT 560	1	1	1	1	1	1	1	1
7220A	テキスト/グラフ 5Mモード	高解像CRT 380	1	1	0	0	1	1	0	0
7220A	グラフ2.5M*2	2236	4	5	4	5	4	5	4	5
7220A	テキスト*2	1120	2	2	2	2	2	2	2	2
7201	通信制御	300	1	1	0	0	1	1	0	0
7210	GPIB	250	1	1	0	0	1	1	0	0

＊1　1tcyは4システムクロック　＊2　スーパーインポーズ時

GDC5MHzモードを使用し，それでも速度が遅いときに，アセンブリ言語でプログラムを組むというのが順番のようです。

アセンブリ言語で記述する場合，連続アクセスに関する制限に注意しなければなりません。

表4-2は，PC-9801の周辺LSIに対して，I/O命令を連続してアクセスする場合に待たなければならない時間を示したものです。たとえば，7220Aのグラフ2.5Mモード標準CRTを見てみましょう。単位は，JMP　$+2命令をいくつ挿入すべきかで表されています。これを見ると，2ないし3個挿入しなければならないのがわかります。パラメータの計算などがあるので，この制限にひっかかることはまずないと思います。しかし，アセンブリ言語でプログラムを組む場合，頭のすみにこの制限があることをいれておいてください。

5 高速化(そのほかの方法)

GDCの描画プログラムを高速化するためのほかの方法を示します。

①不用なコマンドパラメータを送出しない

② GDCが描画中にパラメータの計算を行う

③パラメータの再利用

1 不用なコマンドパラメータを送出しない

たとえば，直線を描画するのに送出しなければならないコマンドは，次のとおりでした。

- TEXTWコマンド
- WRITEコマンド
- CSRWコマンド
- VECTWコマンド
- VECTEコマンド

しかし，ラインパターンやドット修正モードを変更しない場合は

- CSRWコマンド
- VECTWコマンド
- VECTEコマンド

だけで直線を引くことができます。TEXTW，WRITEは初期化時に一度設定するだけでよいのです。不用なコマンド/パラメータの送出は，FIFOがフルになる確率が増し，CPUが待たされる原因になります。また，パラメータの計算などに無駄な時間もかかります。不用なコマンド/パラメータは極力送出しないことです。

2 GDCが描画中にパラメータの計算を行う

GDCにコマンド/パラメータを送出する部分は，次のようになっていました。

```
/*
        コマンド送出関数
*/
void send_command( cmd )
int cmd;
{
        while( (inp(0xa0) & 2) !=0 );
        outp( 0xa2 , cmd );
}

/*
        パラメータ送出関数
*/
void send_parameter( pmtr )
int pmtr;
{
        while( (inp(0xa0) & 2) != 0 );
        outp( 0xa0 , pmtr );
}
```

いずれも，OUTする前にFIFOがFULLでないことを確認しています。

```
while( (inp(0xa0) & 2) != 0 );
```

コマンドやデータの送出は，FIFOがEMPTYであることを確認してもいいわけですが，それではGDCがデータを解釈しないと，CPUは次のデータを送ることができず，遅いプログラムになってしまいます。そこでFIFOがFULLでない間はどんどんデータを送出するようになっているわけです。FIFOがCPUとGDC間のバッファとなって，全体の処理速度の向上に寄与しています。ところで，FIFOがFULLのときは，CPUはただ待っているだけです。この待っている間に，何かほかの処理をすることができれば，プログラムの効率はさらに上がるでしょう。しかし，そういうプログラムは，マルチタスク環境でないと難しいものです。

3 パラメータを覚えておく

GDCの描画プログラムでは，パラメータを計算する部分もかなりあります。描画の際にグラフィックチャージャを使わない場合，描画開始アドレスを変えて各メモ

リプレーンに描画しています。各メモリプレーンを描画するのに使用するパラメータは，CSRWのEAD/WRITEコマンド以外は同一です。プレーン0用のパラメータをプレーン1，プレーン2，プレーン3で再利用するようにすれば，速度が若干向上します。パラメータの計算が複雑な描画であればあるほど，これは顕著でしょう。

「不用なコマンドパラメータを送出しない」，「パラメータの再利用」を考慮した直線描画関数line2( )をリスト7に示します。

◉リスト7

```
/*
        直線描画関数
        (不用なコマンドパラメータを送出しない。)
        (パラメータを再利用)
        GDC2.5MHz用
        グラフィックチャージャ使用せず
*/
void line2(x1,y1,x2,y2,color)
unsigned x1,y1,x2,y2,color;
{
        unsigned ead;
        unsigned dad,dir;
        int temp;
        int dx,dy;
        int csrwp2;
        int vectwp1,vectwp2,vectwp3,vectwp4,vectwp5;

        /* textw */
        send_command( 0x78 );  /* textw command */
        send_word_parameter( 0xffff );  /* pattern */

/* plane 0 */
        /* write */
        if( (color & 1) != 0 )
                send_command( 0x23 );
        else
                send_command( 0x22 );

        /* init */
        if( x1 > x2 ){
                temp=x2; x2=x1; x1=temp;
                temp=y2; y2=y1; y1=temp;
        }
        if( x1 == x2 )
                if( y1 > y2 ){
```

```
                    temp=y2 ; y2=y1 ; y1=temp;
            };

        /* culadr */

        ead = 0x4000+40*y1+x1/16;
        dad = x1 % 16;

        /* csrw */
        csrwp2 = 16*dad;
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( csrwp2 );

        /* caldir */
        dx = x2 - x1;
        dy = y2 - y1;
        if( dy > 0 ){
            if( dy > dx ){
                    dir = 0;
                    temp = dy ; dy = dx ; dx = temp;
            }
            else
                    dir = 1;
        }
        else{
            dy = -dy;
            if( dy < dx )
                    dir = 2;
            else{
                    dir = 3;
                    temp = dy; dy = dx; dx=temp;
            }
        }

        /* culpar */

        vectwp1 = 8+dir;
        vectwp2 = dx;            /* dc */
        vectwp3 = 2*dy - dx;     /* d */
        vectwp4 = 2*(dy-dx);     /* d2*/
        vectwp5 = 2*dy;          /* d1 */
        /* vectw */
        send_command( 0x4c );
        send_parameter( vectwp1 );
        send_word_parameter( vectwp2 );
        send_word_parameter( vectwp3 );
        send_word_parameter( vectwp4 );
```

```
        send_word_parameter( vectwp5 );

        /* vecte */
        send_command( 0x6c );

    /* plane 1 */
        /* write */
        if( (color & 2) != 0 )
                send_command( 0x23 );
        else
                send_command( 0x22 );

        /* csrw */
        ead = ead + 0x4000;
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( csrwp2 );

        /* vectw */
        send_command( 0x4c );
        send_parameter( vectwp1 );
        send_word_parameter( vectwp2 );
        send_word_parameter( vectwp3 );
        send_word_parameter( vectwp4 );
        send_word_parameter( vectwp5 );

        /* vecte */
        send_command( 0x6c );

    /* plane 2 */
        /* write */
        if( (color & 4) != 0 )
                send_command( 0x23 );
        else
                send_command( 0x22 );

        /* csrw */
        ead = ead + 0x4000;
        send_command( 0x49 );
        send_word_parameter( ead );
        send_parameter( csrwp2 );

        /* vectw */
        send_command( 0x4c );
        send_parameter( vectwp1 );
        send_word_parameter( vectwp2 );
        send_word_parameter( vectwp3 );
        send_word_parameter( vectwp4 );
        send_word_parameter( vectwp5 );
```

```
        /* vecte */
        send_command( 0x6c );
}
```

VECTWのパラメータはそっくりそのまま，CSRWのパラメータにも多少加工して再利用しています。また，TEXTWコマンドを最初の1回しか送出していません。これらの工夫による速度向上は5%程度です。

また，描画したデータは消去することが多いものです。描画と消去は，ドット修正モードなどが違うだけで，送出するコマンド/パラメータは同じです。そこで描画するときにパラメータを覚えておき，消去するときはそのパラメータを使うようにすれば，無駄なパラメータ計算を省くことができ，プログラム全体の実行時間を上げることができます。このような細かい注意を払えば，速度は向上します。

高速化についていろいろ解説してきました。効果の高い順番は

①フラッシュ描画の利用

②グラフィックチャージャの利用

③GDC5MHzの利用

④アセンブリ言語の利用

⑤そのほか

- 「不用なコマンドパラメータを送出しない」
- 「GDCが描画中にパラメータの計算を行う」
- 「パラメータの再利用」

となります。グラフィックチャージャとGDC5MHzは手軽に使えるので，これらを利用できる機種(PC-9801VX以上)では使ったほうがよいでしょう。それでも速度が遅い場合は，アセンブリ言語を利用することになります。グラフィックチャージャやGDC5MHzモードも利用していないのに，アセンブリ言語でプログラムを書くことは苦労が多いわりには効果が上がりません。グラフィックチャージャを使っても遅い，GDC5MHzモードを使用してもまだ遅いというときに，アセンブリ言語を使ってください。

第5章 プログラム

GDCの速度を最大限に引き出すためには，やはりアセンブリ言語でプログラムを作成しなければなりません。高速化の章で述べたようにグラフィックチャージャやGDC5MHzモード，フラッシュ描画を使用しても，どうにも間に合わないときはアセンブリ言語で記述すべきでしょう。以下に示すリストは，グラフィックチャージャとGDC5MHzを使用したアセンブリ言語ルーチンです。『Lattice C Compiler』Ver.4.1のスモールモデルから呼び出せる関数になっています。関数は，第3章のものと同じ動作をするようになっています。関数名の頭にmをつけて，機械語ルーチンであることを表しています。さらに，高速化する必要がある人は，メインプログラムもアセンブリ言語で書くしかないでしょう。

1 直線描画関数

まず直線を描画するアセンブリ言語定義の関数を示します。

```
void mline(x1,y1,x2,y2,color)
int x1;          /* X座標1 */
int y1;          /* Y座標1 */
int x2;          /* X座標2 */
int y2;          /* Y座標2 */
int color;       /* 描画色(0≦color≦15) */
```

関数mline( )の定義は，リスト1のようになります。

◉リスト1

```
;
; 直線描画関数
;
;       void mline(x1,y1,x2,y2,color);
;
; 注)   スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;
;       SAVE "MLINE.ASM"

ARGS    STRUC
ARG_X1  DW      ?
ARG_Y1  DW      ?
ARG_X2  DW      ?
ARG_Y2  DW      ?
ARG_COLOR       DW      ?
ARGS    ENDS

DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP
```

```
XX1     DW      0
YY1     DW      0
XX2     DW      0
YY2     DW      0
COLOR0  DW      0

X1      DW      0
Y1      DW      0
X2      DW      0
Y2      DW      0
PARMTR  DB      0
COMMAND DB      0

COLOR   DW      0
EAD     DW      0
DAD     DB      0
DELTA_X DW      0
DELTA_Y DW      0
DIR     DB      0
DC_REG  DW      0
D_REG   DW      0
D1_REG  DW      0
D2_REG  DW      0
WRDPAR  DW      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0

_DATA   ENDS
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT
        ASSUME  DS:DGROUP

STATPT  EQU     0A0H
POUTPT  EQU     0A0H
COUTPT  EQU     0A2H

        PUBLIC  MLINE
MLINE   PROC    NEAR
        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_X1
        MOV     X1,AX
        MOV     AX,[BP+4].ARG_Y1
        MOV     Y1,AX
```

```
        MOV     AX,[BP+4].ARG_X2
        MOV     X2,AX
        MOV     AX,[BP+4].ARG_Y2
        MOV     Y2,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM

TEXTW:  MOV     AL,78H
        MOV     COMMAND,AL
        CALL    OUTC
        MOV     WRDPAR,0FFFFH
        CALL    OUTWP
WRITE:
        MOV     AL,20H                  ; 20H REPLACE COMMAND
        MOV     COMMAND,AL
        CALL    OUTC
INIT:                                   ; LET DELTA_X=X2-X1>=0
        MOV     AX,X2
        CMP     AX,X1
        JGE     INIT1                   ; IF DELTA_X>=0 THEN INIT1

        MOV     BX,X1                   ; SWAP X1,X2
        MOV     X1,AX
        MOV     X2,BX
        MOV     AX,Y1                   ; SWAP Y1,Y2
        MOV     BX,Y2
        MOV     Y2,AX
        MOV     Y1,BX
INIT1:  MOV     AX,X1
        CMP     AX,X2
        JNE     INIT2                   ; IF X1<>X2 THEN INIT2

        MOV     AX,Y2
        CMP     AX,Y1
        JGE     INIT3                   ; IF Y2>=Y1 THEN INIT3

        MOV     BX,Y1                   ; SWAP Y1,Y2
        MOV     Y1,AX
        MOV     Y2,BX

INIT2:
INIT3:
```

```
CULADR:
        MOV     CX,4000H            ; LET V-RAM START ADRESS=4000H
        MOV     AX,X1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1                ; AX=INT(X1/16)

        ADD     CX,AX

        MOV     AX,Y1
        MOV     BX,AX
        SHL     BX,1
        SHL     BX,1
        ADD     AX,BX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1                ; AX=40*Y1

        ADD     CX,AX
        MOV     EAD,CX

        MOV     AX,X1
        AND     AX,0FH              ; AX=X1 MOD 16
        MOV     DAD,AL
;
CSRW:
        MOV     AL,49H              ; CSRW COMMAND=49H
        MOV     COMMAND,AL
        CALL    OUTC

        MOV     AX,EAD
        MOV     PARMTR,AL           ; COMMAND OUT EADL
        CALL    OUTP
        MOV     PARMTR,AH           ; COMMAND OUT EADH
        CALL    OUTP

        MOV     AL,DAD
        MOV     CL,4
        SHL     AL,CL               ; SHIFT DAD 4 TIMES
        MOV     PARMTR,AL
        CALL    OUTP
;
CULDIR:
        MOV     AX,X2
        SUB     AX,X1
        MOV     DELTA_X,AX                  ; DELTA_X=X2-X1
```

```
        MOV     AX,Y2
        SUB     AX,Y1
        MOV     DELTA_Y,AX                      ; DELTA_Y=Y2-Y1

        MOV     AX,DELTA_Y
        CMP     AX,0
        JLE     DIR1            ; IF DELTA_Y<=0 THEN DIR1

        MOV     BX,DELTA_X
        CMP     AX,BX
        JLE     DIR2

        XOR     AL,AL
        MOV     DIR,AL          ; DIR=0

        MOV     AX,DELTA_X
        MOV     BX,DELTA_Y
        MOV     DELTA_Y,AX
        MOV     DELTA_X,BX                      ; SWAP DELTA_X,DELTA_Y

        JMP     DIR3

DIR2:   MOV     AL,1
        MOV     DIR,AL          ; DIR=1
        JMP     DIR3

DIR1:   MOV     BX,DELTA_Y
        NEG     BX
        MOV     DELTA_Y,BX                      ; DELTA_Y=ABS(DELTA_Y)

        MOV     AX,DELTA_X
        CMP     BX,AX
        JG      DIR4            ; IF DELTA_Y>DELTA_X THEN DIR4

        MOV     AL,2
        MOV     DIR,AL          ; DIR=2
        JMP     DIR3

DIR4:   MOV     AL,3
        MOV     DIR,AL

        MOV     AX,DELTA_X
        MOV     BX,DELTA_Y
        MOV     DELTA_Y,AX
        MOV     DELTA_X,BX                      ; SWAP DELTA_X,DELTA_Y

DIR3:
;
```

```
CULPAR:
        MOV     AX,DELTA_X
        MOV     DC_REG,AX           ; LET DC_REG=DELTA_X

        MOV     AX,DELTA_Y
        SAL     AX,1
        SAL     AX,1
        SUB     AX,DELTA_X
        MOV     D_REG,AX            ; LET D_REG=2*DELTA_Y-DELTA_X

        MOV     AX,DELTA_Y
        SUB     AX,DELTA_X
        SAL     AX,1
        SAL     AX,1
        MOV     D2_REG,AX           ; LET D2_REG=2*DELTA_Y-2*DELTA_X

        MOV     AX,DELTA_Y

        SAL     AX,1
        SAL     AX,1
        MOV     D1_REG,AX           ; LET D1_REG=2*DELTA_Y

;
VECTW:
        MOV     AL,4CH              ; VECTW COMMAND=4CH
        MOV     COMMAND,AL
        CALL    OUTC

        MOV     AL,DIR
        OR      AL,8                ; DECLEAR TO DRAW 'LINE'
        MOV     PARMTR,AL
        CALL    OUTP

        MOV     AX,DC_REG
        OR      AH,64               ; DGD BIT ON
        MOV     WRDPAR,AX
        CALL    OUTWP

        MOV     AX,D_REG
        MOV     WRDPAR,AX
        CALL    OUTWP

        MOV     AX,D2_REG
        MOV     WRDPAR,AX
        CALL    OUTWP

        MOV     AX,D1_REG
        MOV     WRDPAR,AX
        CALL    OUTWP

;
```

```
    ; SET GRAPHIC CHARGER
    ;
    GC1:      IN        AL, 0A0H
              TEST      AL, 8
              JNZ       GC1

              MOV       AL, 0C0H
              OUT       07CH, AL

              MOV       TILE0, 0
              MOV       TILE1, 0
              MOV       TILE2, 0
              MOV       TILE3, 0

              TEST      COLOR, 1
              JZ        M1
              MOV       TILE0, 0FFH
    M1:
              TEST      COLOR, 2
              JZ        M2
              MOV       TILE1, 0FFH
    M2:
              TEST      COLOR, 4
              JZ        M3
              MOV       TILE2, 0FFH
    M3:
              TEST      COLOR, 8
              JZ        M4
              MOV       TILE3, 0FFH
    M4:
              MOV       AL, TILE0
              OUT       07EH, AL
              MOV       AL, TILE1
              OUT       07EH, AL
              MOV       AL, TILE2
              OUT       07EH, AL
              MOV       AL, TILE3
              OUT       07EH, AL
    VECTE:
              MOV       AL, 6CH                 ; VECTE COMMAND=6CH
              MOV       COMMAND, AL
              CALL      OUTC

              IRP       REG, <ES, DS, DI, SI, DX, CX, BX>
              POP       REG
              ENDM

              MOV       SP, BP
              POP       BP
```

```
        RET
;
;       END OF LINE ROUTINE
        ;
;
;       COMMAND OUT ROUTINE FOR GDC
;
OUTC:
;
OUTCLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTCLP                  ; WAIT UNTIL FIFO NOT FULL
;
        MOV     AL,COMMAND
        OUT     COUTPT,AL
;
        RET
;
;       PARAMETER OUT ROUTINE FOR GDC
;
OUTP:
;
OUTPLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTPLP                  ; WAIT UNTIL FIFO NOT FULL
;
        MOV     AL,PARMTR
        OUT     POUTPT,AL
;
        RET
;
;
OUTWP:
        MOV     AX,WRDPAR
        MOV     PARMTR,AL
        CALL    OUTP
        MOV     PARMTR,AH
        CALL    OUTP

        RET
MLINE   ENDP
_TEXT   ENDS
        END
```

2
円描画関数

円を描画するアセンブリ言語定義の関数mcircle()を示します。

```
void mcircle(x,y,r,color)
int x;          /* X座標 */
int y;          /* Y座標 */
int r;          /* 半径 */
int color;      /* 描画色(0≦color≦15) */
```

関数mcircle()の定義はリスト2のようになります。

◉リスト2

```
;   円描画関数
;       void mcircle(x,y,radius,color);
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;
;       SAVE "MCIRCLE.ASM"

ARGS     STRUC
ARG_X    DW      ?
ARG_Y    DW      ?
ARG_RADIUS       DW      ?
ARG_COLOR        DW      ?
ARGS     ENDS

DGROUP   GROUP   _DATA
_DATA    SEGMENT WORD PUBLIC 'DATA'
         ASSUME  DS:DGROUP

X_C      DW      0
Y_C      DW      0
```

```
RADIUS  DW      0
X_CRD   DW      0
Y_CRD   DW      0
COMMAND DB      0
PARMTR  DB      0
COLOR   DW      0
EAD     DW      0
DAD     DB      0
DIR     DB      0

DC_REG  DW      0
D_REG   DW      0
D2_REG  DW      0
D1_REG  DW      0
DM_REG  DW      0
WRDPAR  DW      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0

_DATA   ENDS

_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
STATPT  EQU     0A0H

        PUBLIC  MCIRCLE
MCIRCLE PROC    NEAR

        PUSH    BP
        MOV     BP,SP
        MOV     AX,[BP+4].ARG_X
        MOV     X_C,AX
        MOV     AX,[BP+4].ARG_Y
        MOV     Y_C,AX
        MOV     AX,[BP+4].ARG_RADIUS
        MOV     RADIUS,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH  REG
        ENDM
```

```
        CALL    TEXTW
        CALL    WRITE
        CALL    SET_VECTW_DATA
        CALL    SET_GC
        CALL    ARC74
        CALL    ARC16
        CALL    ARC30
        CALL    ARC52

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET

TEXTW:
        MOV     COMMAND,78H
        CALL    OUTC
        MOV     WRDPAR,0FFFFH
        CALL    OUTWP
        RET

WRITE:
        MOV     COMMAND,20H
        CALL    OUTC
        RET

SET_VECTW_DATA:
        MOV     AX,RADIUS
        MOV     BX,10000*3
        MUL     BX
        MOV     BX,14142*3
        DIV     BX
        INC     AX
        OR      AH,64               ; DGD BIT ON
        MOV     DC_REG,AX

        MOV     AX,RADIUS
        DEC     AX
        MOV     D_REG,AX
        SAL     AX,1
        MOV     D2_REG,AX

        MOV     D1_REG,-1
        MOV     DM_REG,0
        RET
```

```
SET_GC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0

        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL
        RET

ARC74:
        MOV     AX,X_C
        ADD     AX,RADIUS
        MOV     X_CRD,AX

        MOV     AX,Y_C
        MOV     Y_CRD,AX
```

```
        CALL    CSRW
        MOV     DIR,7
        CALL    VECTW
        CALL    VECTE

        CALL    CSRW
        MOV     DIR,4
        CALL    VECTW
        CALL    VECTE
        RET
ARC16:
        MOV     AX,X_C
        MOV     X_CRD,AX

        MOV     AX,Y_C
        SUB     AX,RADIUS
        MOV     Y_CRD,AX

        CALL    CSRW
        MOV     DIR,1
        CALL    VECTW
        CALL    VECTE

        CALL    CSRW
        MOV     DIR,6
        CALL    VECTW
        CALL    VECTE
        RET
ARC30:
        MOV     AX,X_C
        SUB     AX,RADIUS
        MOV     X_CRD,AX

        MOV     AX,Y_C
        MOV     Y_CRD,AX

        CALL    CSRW
        MOV     DIR,3
        CALL    VECTW
        CALL    VECTE

        CALL    CSRW
        MOV     DIR,0
        CALL    VECTW
        CALL    VECTE
        RET
```

```
ARC52:
        MOV     AX,X_C
        MOV     X_CRD,AX

        MOV     AX,Y_C
        ADD     AX,RADIUS
        MOV     Y_CRD,AX

        CALL    CSRW
        MOV     DIR,5
        CALL    VECTW
        CALL    VECTE

        CALL    CSRW
        MOV     DIR,2
        CALL    VECTW
        CALL    VECTE
        RET

CSRW:
        MOV     AX,Y_CRD
        MOV     CX,AX
        SHL     CX,1
        SHL     CX,1
        ADD     AX,CX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1                    ; AX=40*Y_CRD

        ADD     AX,04000H
        MOV     BX,X_CRD

        SAR     BX,1
        SAR     BX,1
        SAR     BX,1
        SAR     BX,1

        ADD     AX,BX
        MOV     EAD,AX
        MOV     AX,X_CRD
        AND     AL,15
        MOV     DAD,AL

        MOV     COMMAND,49H
        CALL    OUTC

        MOV     AX,EAD
        MOV     WRDPAR,AX
        CALL    OUTWP
```

```
        MOV     AL,DAD
        SAL     AL,1
        SAL     AL,1
        SAL     AL,1
        SAL     AL,1
        MOV     PARMTR,AL
        CALL    OUTP
        RET

VECTW:
        MOV     COMMAND,4CH
        CALL    OUTC
        MOV     AL,20H
        ADD     AL,DIR
        MOV     PARMTR,AL
        CALL    OUTP

        MOV     AX,DC_REG
        MOV     WRDPAR,AX

        CALL    OUTWP

        MOV     AX,D_REG
        MOV     WRDPAR,AX
        CALL    OUTWP

        MOV     AX,D2_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,D1_REG
        MOV     WRDPAR,AX
        CALL    OUTWP

        MOV     AX,DM_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        RET

VECTE:
        MOV     COMMAND,6CH
        CALL    OUTC
        RET
OUTC:
OUTCLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTCLP
        MOV     AL,COMMAND
        OUT     COUTPT,AL
        RET
```

```
OUTP:
OUTPLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTPLP

        MOV     AL,PARMTR
        OUT     POUTPT,AL
        RET

OUTWP:
        MOV     AX,WRDPAR
        MOV     PARMTR,AL
        CALL    OUTP
        MOV     PARMTR,AH
        CALL    OUTP
        RET

MCIRCLE ENDP

_TEXT   ENDS
        END
```

3 四辺形描画関数

四辺形を描画するアセンブリ言語定義の関数mbox()を示します。

```
void mbox(x1, y1, x2, y2, color)
int x1;        /* X座標1 */
int y1;        /* Y座標1 */
int x2;        /* X座標2 */
int y2;        /* Y座標2 */
int color;     /* 描画色(0≦color≦15) */
```

関数mbox()の定義はリスト3のようになります。

◉リスト3

```
;
;  四辺形描画関数
;        void mbox(x1, y1, x2, y2, color);
;
;        スモールモデル
;        GDC  5MHz
;        グラフィックチャージャ使用
;
;

;
; スタック上の引数を表わす
; 構造体
;
ARGS     STRUC
ARG_X1   DW        ?
ARG_Y1   DW        ?
ARG_X2   DW        ?
ARG_Y2   DW        ?
ARG_COLOR          DW        ?
ARGS     ENDS
```

```
;
; データセグメント
;
DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP

X1      DW      0
Y1      DW      0
X2      DW      0
Y2      DW      0

DELTA_X DW      0
DELTA_Y DW      0
COLOR   DW      0

DC_REG  DW      0
D_REG   DW      0
D2_REG  DW      0
D1_REG  DW      0
DM_REG  DW      0

EAD     DW      0
DAD     DB      0
COMMAND DB      0
PARMTR  DB      0
WRDPAR  DW      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0

_DATA   ENDS

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
STATPT  EQU     0A0H

        PUBLIC  MBOX

MBOX    PROC    NEAR
```

```
        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_X1
        MOV     X1,AX
        MOV     AX,[BP+4].ARG_Y1
        MOV     Y1,AX
        MOV     AX,[BP+4].ARG_X2
        MOV     X2,AX
        MOV     AX,[BP+4].ARG_Y2
        MOV     Y2,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM
        CALL    TEXTW
        CALL    WRITE
        CALL    SET_DX_AND_DY
        CALL    CUL_EAD_AND_DAD
        CALL    CSRW
        CALL    SET_DRAWING_PRAMETER
        CALL    VECTW
        CALL    SET_GC
        CALL    VECTE

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET

TEXTW:
        MOV     COMMAND,78H
        CALL    OUTC
        MOV     WRDPAR,0FFFFH
        CALL    OUTWP
        RET

WRITE:
        MOV     COMMAND,20H
        CALL    OUTC
        RET

SET_DX_AND_DY:
        MOV     AX,X1
```

```
		CMP	AX,X2
		JLE	SET1
		XCHG	X2,AX
		MOV	X1,AX
SET1:
		MOV	AX,Y1
		CMP	AX,Y2
		JLE	SET2
		XCHG	Y2,AX
		MOV	Y1,AX
SET2:
		MOV	AX,X2
		SUB	AX,X1
		MOV	DELTA_X,AX
		MOV	AX,Y2
		SUB	AX,Y1
		MOV	DELTA_Y,AX
		RET

SET_DRAWING_PRAMETER:
		MOV	DC_REG,3
		MOV	AX,DELTA_Y
		MOV	D_REG,AX
		MOV	AX,DELTA_X
		MOV	D2_REG,AX
		MOV	D1_REG,-1
		MOV	AX,DELTA_Y
		MOV	DM_REG,AX
		RET

CUL_EAD_AND_DAD:
		MOV	BX,4000H

		MOV	AX,X1
		SHR	AX,1
		SHR	AX,1
		SHR	AX,1
		SHR	AX,1

		ADD	BX,AX

		MOV	AX,Y1
		MOV	CX,AX
		SHL	CX,1
		SHL	CX,1
		ADD	AX,CX
		SHL	AX,1
		SHL	AX,1
		SHL	AX,1
```

```
        ADD     BX,AX
        MOV     EAD,BX
        MOV     AX,X1
        AND     AL,15
        MOV     DAD,AL
        RET

CSRW:
        MOV     COMMAND,49H
        CALL    OUTC
        MOV     AX,EAD
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AL,DAD
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1

        MOV     PARMTR,AL
        CALL    OUTP
        RET
VECTW:
        MOV     COMMAND,4CH
        CALL    OUTC
        MOV     PARMTR,40H
        CALL    OUTP
        MOV     AX,DC_REG
        OR      AH,64                   ; DGD BIT ON
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,D_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,D2_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,D1_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,DM_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        RET

VECTE:
        MOV     COMMAND,6CH
        CALL    OUTC
        RET
```

```
SET_GC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0

        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL
        RET

;
; コマンド送出ルーチン
;
OUTC:
OUTCLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTCLP
```

```
        MOV     AL,COMMAND
        OUT     COUTPT,AL
        RET

;
; パラメータ送出ルーチン
;
OUTP:
OUTPLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTPLP

        MOV     AL,PARMTR
        OUT     POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
OUTWP:
        MOV     AX,WRDPAR
        MOV     PARMTR,AL
        CALL    OUTP
        MOV     PARMTR,AH
        CALL    OUTP
        RET

MBOX    ENDP
_TEXT   ENDS
        END
```

4 円弧描画関数

アセンブリ言語で定義した円弧描画関数marc()を示します。

```
void marc(x,y,r,color,start_angle,end_angle);
int x;           /* 中心X座標1 */
int y;           /* 中心Y座標1 */
int r;           /* 半径 */
int color;       /* 描画色(0≦color≦15) */
int start_angle;/* 開始角度(度の10倍  0以上3599以下 */
int end_angle;  /* 終了角度(度の10倍  0以上3599以下 */
```

関数marc()の定義は，リスト4のようになります。

◉リスト4　"marc.c"

```
;
; 円弧描画関数(アセンブリ言語定義)
;
;       void marc(x,y,r,color,start,end);
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用

;
; スタック上の引数を表わす
; 構造体
;
ARGS    STRUC
ARG_X   DW      ?
ARG_Y   DW      ?
ARG_R   DW      ?
ARG_COLOR       DW      ?
ARG_START       DW      ?
ARG_END         DW      ?
ARGS    ENDS
```

```
;
; データセグメント
;
DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP
        PUBLIC  X_C,Y_C,RADIUS,ALPHA,BETA,DIR,DC_REG,D_REG,D1_REG,D2_REG
        PUBLIC  DM_REG,EAD,DAD,XCRD,YCRD,I,J,N
        PUBLIC  A,B,AA,BB,IA,IB,START_ANGLE,END_ANGLE,COLOR
        PUBLIC  TILE0,TILE1,TILE2,TILE3

X_C     DW      0
Y_C     DW      0
RADIUS  DW      0
ALPHA   DW      0
BETA    DW      0
DIR     DW      0

DC_REG  DW      0
D_REG   DW      0
D1_REG  DW      0
D2_REG  DW      0
DM_REG  DW      0

EAD     DW      0
DAD     DW      0

XCRD    DW      0
YCRD    DW      0

I       DW      0
J       DW      0
N       DW      0

A       DW      0
B       DW      0
AA      DW      0
BB      DW      0

IA      DW      0
IB      DW      0
II      DW      0

START_ANGLE     DW      0
END_ANGLE       DW      0

COLOR   DW      0
TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0
```

```
SINTABLE        DW      0,   1,   3,   5,   6,   8,  10,  12,  13,  15
                DW     17,  19,  20,  22,  24,  26,  27,  29,  31,  33
                DW     34,  36,  38,  40,  41,  43,  45,  47,  48,  50
                DW     52,  54,  55,  57,  59,  61,  62,  64,  66,  68
                DW     69,  71,  73,  74,  76,  78,  80,  81,  83,  85
                DW     87,  88,  90,  92,  94,  95,  97,  99, 101, 102
                DW    104, 106, 107, 109, 111, 113, 114, 116, 118, 120
                DW    121, 123, 125, 127, 128, 130, 132, 133, 135, 137
                DW    139, 140, 142, 144, 146, 147, 149, 151, 152, 154
                DW    156, 158, 159, 161, 163, 165, 166, 168, 170, 171
                DW    173, 175, 177, 178, 180, 182, 183, 185, 187, 189
                DW    190, 192, 194, 195, 197, 199, 201, 202, 204, 206
                DW    207, 209, 211, 213, 214, 216, 218, 219, 221, 223
                DW    224, 226, 228, 230, 231, 233, 235, 236, 238, 240
                DW    241, 243, 245, 246, 248, 250, 252, 253, 255, 257
                DW    258, 260, 262, 263, 265, 267, 268, 270, 272, 273
                DW    275, 277, 278, 280, 282, 284, 285, 287, 289, 290
                DW    292, 294, 295, 297, 299, 300, 302, 304, 305, 307
                DW    309, 310, 312, 313, 315, 317, 318, 320, 322, 323
                DW    325, 327, 328, 330, 332, 333, 335, 337, 338, 340
                DW    342, 343, 345, 346, 348, 350, 351, 353, 355, 356
                DW    358, 359, 361, 363, 364, 366, 368, 369, 371, 372
                DW    374, 376, 377, 379, 381, 382, 384, 385, 387, 389
                DW    390, 392, 393, 395, 397, 398, 400, 401, 403, 405
                DW    406, 408, 409, 411, 413, 414, 416, 417, 419, 421
                DW    422, 424, 425, 427, 428, 430, 432, 433, 435, 436
                DW    438, 439, 441, 443, 444, 446, 447, 449, 450, 452
                DW    453, 455, 457, 458, 460, 461, 463, 464, 466, 467
                DW    469, 471, 472, 474, 475, 477, 478, 480, 481, 483
                DW    484, 486, 487, 489, 490, 492, 493, 495, 496, 498
                DW    500, 501, 503, 504, 506, 507, 509, 510, 512, 513
                DW    515, 516, 518, 519, 521, 522, 523, 525, 526, 528
                DW    529, 531, 532, 534, 535, 537, 538, 540, 541, 543
                DW    544, 546, 547, 549, 550, 551, 553, 554, 556, 557
                DW    559, 560, 562, 563, 564, 566, 567, 569, 570, 572
                DW    573, 575, 576, 577, 579, 580, 582, 583, 584, 586
                DW    587, 589, 590, 592, 593, 594, 596, 597, 599, 600
                DW    601, 603, 604, 605, 607, 608, 610, 611, 612, 614
                DW    615, 617, 618, 619, 621, 622, 623, 625, 626, 627
                DW    629, 630, 632, 633, 634, 636, 637, 638, 640, 641
                DW    642, 644, 645, 646, 648, 649, 650, 652, 653, 654
                DW    656, 657, 658, 660, 661, 662, 663, 665, 666, 667
                DW    669, 670, 671, 673, 674, 675, 676, 678, 679, 680
                DW    681, 683, 684, 685, 687, 688, 689, 690, 692, 693
                DW    694, 695, 697, 698, 699, 700, 702, 703, 704, 705
                DW    707, 708, 709, 710, 712, 713, 714, 715, 716, 718
                DW    719, 720, 721, 722, 724, 725, 726, 727, 728, 730
                DW    731, 732, 733, 734, 736, 737, 738, 739, 740, 741
                DW    743, 744, 745, 746, 747, 748, 750, 751, 752, 753
                DW    754, 755, 756, 758, 759, 760, 761, 762, 763, 764
                DW    766, 767, 768, 769, 770, 771, 772, 773, 774, 776
                DW    777, 778, 779, 780, 781, 782, 783, 784, 785, 786
```

```
        DW      788, 789, 790, 791, 792, 793, 794, 795, 796, 797
        DW      798, 799, 800, 801, 802, 803, 804, 805, 806, 807
        DW      809, 810, 811, 812, 813, 814, 815, 816, 817, 818
        DW      819, 820, 821, 822, 823, 824, 825, 826, 827, 828
        DW      829, 830, 830, 831, 832, 833, 834, 835, 836, 837
        DW      838, 839, 840, 841, 842, 843, 844, 845, 846, 847
        DW      848, 848, 849, 850, 851, 852, 853, 854, 855, 856
        DW      857, 858, 858, 859, 860, 861, 862, 863, 864, 865
        DW      866, 866, 867, 868, 869, 870, 871, 872, 872, 873
        DW      874, 875, 876, 877, 877, 878, 879, 880, 881, 882
        DW      882, 883, 884, 885, 886, 887, 887, 888, 889, 890
        DW      891, 891, 892, 893, 894, 894, 895, 896, 897, 898
        DW      898, 899, 900, 901, 901, 902, 903, 904, 904, 905
        DW      906, 907, 907, 908, 909, 909, 910, 911, 912, 912
        DW      913, 914, 914, 915, 916, 917, 917, 918, 919, 919
        DW      920, 921, 921, 922, 923, 923, 924, 925, 925, 926
        DW      927, 927, 928, 929, 929, 930, 931, 931, 932, 932
        DW      933, 934, 934, 935, 936, 936, 937, 937, 938, 939
        DW      939, 940, 940, 941, 942, 942, 943, 943, 944, 944
        DW      945, 946, 946, 947, 947, 948, 948, 949, 949, 950
        DW      951, 951, 952, 952, 953, 953, 954, 954, 955, 955
        DW      956, 956, 957, 957, 958, 958, 959, 959, 960, 960
        DW      961, 961, 962, 962, 963, 963, 964, 964, 965, 965
        DW      965, 966, 966, 967, 967, 968, 968, 969, 969, 969
        DW      970, 970, 971, 971, 971, 972, 972, 973, 973, 973
        DW      974, 974, 975, 975, 975, 976, 976, 977, 977, 977
        DW      978, 978, 978, 979, 979, 979, 980, 980, 980, 981
        DW      981, 981, 982, 982, 982, 983, 983, 983, 984, 984
        DW      984, 985, 985, 985, 985, 986, 986, 986, 987, 987
        DW      987, 987, 988, 988, 988, 989, 989, 989, 989, 990
        DW      990, 990, 990, 990, 991, 991, 991, 991, 992, 992
        DW      992, 992, 992, 993, 993, 993, 993, 993, 994, 994
        DW      994, 994, 994, 995, 995, 995, 995, 995, 995, 996
        DW      996, 996, 996, 996, 996, 996, 997, 997, 997, 997
        DW      997, 997, 997, 997, 998, 998, 998, 998, 998, 998
        DW      998, 998, 998, 998, 998, 999, 999, 999, 999, 999
        DW      999, 999, 999, 999, 999, 999, 999, 999, 999, 999
        DW      999, 999, 999, 999, 999, 999, 999, 999, 999, 999
        DW      1000
_DATA   ENDS
;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT
;
; 定数
;
COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
```

```
STATPT  EQU     0A0H

        PUBLIC  MARC

MARC    PROC    NEAR
        PUSH    BP
        MOV     BP, SP

        MOV     AX, [BP+4].ARG_X
        MOV     X_C, AX
        MOV     AX, [BP+4].ARG_Y
        MOV     Y_C, AX
        MOV     AX, [BP+4].ARG_R
        MOV     RADIUS, AX
        MOV     AX, [BP+4].ARG_COLOR
        MOV     COLOR, AX
        MOV     AX, [BP+4].ARG_START
        MOV     START_ANGLE, AX
        MOV     AX, [BP+4].ARG_END
        MOV     END_ANGLE, AX

        IRP     REG, <BX, CX, DX, SI, DI, DS, ES>
                PUSH    REG
        ENDM

        CALL    ARC

        IRP     REG, <ES, DS, DI, SI, DX, CX, BX>
        POP     REG
        ENDM

        MOV     SP, BP
        POP     BP
        RET
ARC00:
        MOV     ALPHA, AX
        MOV     BETA, BX
        MOV     DIR, CX

        MOV     AX, BETA
        CALL    ISIN
        IMUL    RADIUS
        MOV     BX, 1000
        IDIV    BX
        OR      AH, 64              ; DGD bit ON
        MOV     DC_REG, AX
```

```
        MOV     AX,RADIUS
        DEC     AX
        MOV     D_REG,AX

        MOV     AX,D_REG
        SHL     AX,1
        MOV     D2_REG,AX

        MOV     D1_REG,-1

        MOV     AX,ALPHA
        CALL    ISIN
        IMUL    RADIUS
        MOV     BX,1000
        IDIV    BX
        MOV     DM_REG,AX

        CMP     DIR,7
        JE      L01
        CMP     DIR,4
        JE      L01
        JMP     L1
L01:
        MOV     AX,X_C
        ADD     AX,RADIUS
        MOV     XCRD,AX

        MOV     AX,Y_C
        MOV     YCRD,AX

L1:
        CMP     DIR,0
        JE      L02
        CMP     DIR,3
        JE      L02
        JMP     L2
L02:
        MOV     AX,X_C
        SUB     AX,RADIUS
        MOV     XCRD,AX

        MOV     AX,Y_C
        MOV     YCRD,AX

L2:
        CMP     DIR,1
        JE      L03
        CMP     DIR,6
```

```
        JE      L03
        JMP     L3
L03:
        MOV     AX,X_C
        MOV     XCRD,AX

        MOV     AX,Y_C
        SUB     AX,RADIUS
        MOV     YCRD,AX

L3:
        CMP     DIR,2
        JE      L04
        CMP     DIR,5
        JE      L04
        JMP     L4
L04:
        MOV     AX,X_C
        MOV     XCRD,AX

        MOV     AX,Y_C
        ADD     AX,RADIUS
        MOV     YCRD,AX
L4:
        MOV     AX,YCRD
        MOV     BX,AX
        SHL     BX,1
        SHL     BX,1
        ADD     AX,BX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1
        MOV     CX,AX

        MOV     AX,XCRD
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        ADD     AX,CX
        MOV     EAD,AX

        MOV     AX,XCRD
        AND     AX,000FH
        MOV     DAD,AX

        PUBLIC  CSRW
```

```
CSRW:
        MOV     AX,49H
        CALL    SENDC

        MOV     AX,EAD
        CALL    SENDWP

        MOV     AX,DAD
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1
        CALL    SENDP

        MOV     AX,4CH
        CALL    SENDC

        MOV     AX,20H
        ADD     AX,DIR
        CALL    SENDP

        MOV     AX,DC_REG
        CALL    SENDWP

        MOV     AX,D_REG
        CALL    SENDWP

        MOV     AX,D2_REG
        CALL    SENDWP

        MOV     AX,D1_REG
        CALL    SENDWP

        MOV     AX,DM_REG
        CALL    SENDWP

        MOV     AX,6CH
        CALL    SENDC

        RET

ARC0_INTERNAL:
        MOV     A,AX
        MOV     B,BX
        MOV     II,CX

        MOV     AX,II
        MOV     BX,AX
```

```
        SHL     BX,1
        ADD     AX,BX
        ADD     AX,7
        AND     AX,7
        MOV     DIR,AX

        MOV     AX,II
        TEST    AX,1
        JNZ     I1

        MOV     AX,A
        MOV     BX,B
        MOV     CX,DIR
        CALL    ARC00
        JMP     I2

I1:
        MOV     AX,450
        SUB     AX,B
        MOV     BX,450
        SUB     BX,A
        MOV     CX,DIR
        CALL    ARC00
I2:
        RET


ARC0:
        MOV     AX,78H
        CALL    SENDC

        MOV     AX,0FFFFH
        CALL    SENDWP

        MOV     AX,START_ANGLE
        CWD
        MOV     BX,450
        IDIV    BX
        MOV     IA,AX
        MOV     AA,DX

        MOV     AX,END_ANGLE
        CWD
        MOV     BX,450
        IDIV    BX
        MOV     IB,AX
        MOV     BB,DX
```

```
        MOV     AX,IB
        SUB     AX,IA
        ADD     AX,7
        CWD
        MOV     BX,8
        IDIV    BX
        MOV     N,DX

        MOV     AX,IA
        CMP     AX,IB
        JNE     A1

        MOV     AX,START_ANGLE
        CMP     AX,END_ANGLE
        JG      A1

        MOV     AX,AA
        MOV     BX,BB
        MOV     CX,IA
        CALL    ARC0_INTERNAL
        JMP     A2
A1:
        MOV     AX,AA
        MOV     BX,450
        MOV     CX,IA
        CALL    ARC0_INTERNAL

        MOV     J,1
FOR1:
        MOV     AX,J
        CMP     AX,N
        JG      NEXT1

        MOV     AX,IA
        ADD     AX,J
        AND     AX,7
        MOV     I,AX

        MOV     AX,0
        MOV     BX,450
        MOV     CX,I
        CALL    ARC0_INTERNAL
        INC     J
        JMP     FOR1
NEXT1:
        MOV     AX,0
        MOV     BX,BB
        MOV     CX,IB
```

```
        CALL    ARC0_INTERNAL
A2:
        RET

ARC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0

        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL

; WRITE COMMAND
        MOV     AX,023H
        CALL    SENDC
```

```
        CALL    ARCO
        RET
;
; コマンド送出ルーチン
;
SENDC:
        MOV     BX,AX
SENDCLP:        IN       AL,STATPT
        AND     AL,2
        JNZ     SENDCLP
        MOV     AX,BX
        OUT     COUTPT,AL
        RET
;
; パラメータ送出ルーチン
;
SENDP:
        MOV     BX,AX
SENDPLP:        IN       AL,STATPT
        AND     AL,2
        JNZ     SENDPLP
        MOV     AX,BX
        OUT     POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
SENDWP:
        MOV     CX,AX
        CALL    SENDP
        MOV     AL,CH
        CALL    SENDP
        RET

ISIN:
        CWD
        MOV     BX,3600
        IDIV    BX
        MOV     DI,0            ; SIGN
        CMP     DX,0
        JGE     S1
        MOV     DI,0FFFFH       ; MINUS
        NEG     DX
S1:
        CMP     DX,900
        JG      S2
```

```
        MOV     SI,DX
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        JMP     SINEND
S2:
        CMP     DX,1800
        JG      S3
        MOV     SI,1800
        SUB     SI,DX
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        JMP     SINEND
S3:
        CMP     DX,2700
        JG      S4
        MOV     SI,DX
        SUB     SI,1800
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        NEG     AX
        JMP     SINEND
S4:
        MOV     SI,3600
        SUB     SI,DX
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        NEG     AX
SINEND:
        CMP     DI,0
        JNE     S5
        RET
S5:
        NEG     AX
        RET

ICOS:
        ADD     AX,900
        CALL    ISIN
        RET

MARC    ENDP
 TEXT   ENDS
        END
```

5 傾斜四辺形描画関数

傾斜四辺形を描画する，アセンブリ言語定義の関数mbox2()を示します。

```
void mbox2(x1,y1,x2,y2,color)
int x1;           /* X座標1  */
int y1;           /* Y座標1  */
int x2;           /* X座標2  */
int y2;           /* Y座標2  */
int color;        /* 描画色(0≦color≦15) */
```

関数mbox2()の定義は，リスト5のようになります。

◉リスト5　"mbox2.c"

```
;
;  傾斜四辺形描画関数(アセンブリ言語定義)
;
;       void mbox2(x1,y1,x2,y2,color);
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;

;
; スタック上の引数を表わす
; 構造体
;
ARGS    STRUC
ARG_X1  DW      ?
ARG_Y1  DW      ?
ARG_X2  DW      ?
ARG_Y2  DW      ?
ARG_COLOR       DW      ?
ARGS    ENDS
```

```
;
; データセグメント
;
DGROUP  GROUP    _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP

X1      DW      0
Y1      DW      0
X2      DW      0
Y2      DW      0

DELTA_X DW      0
DELTA_Y DW      0
COLOR   DW      0

DC_REG  DW      0
D_REG   DW      0
D2_REG  DW      0
D1_REG  DW      0
DM_REG  DW      0

EAD     DW      0
DAD     DB      0
COMMAND DB      0
PARMTR  DB      0
WRDPAR  DW      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0

A       DW      0
B       DW      0
_DATA   ENDS

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
STATPT  EQU     0A0H

ABS     MACRO   REG
```

```
        LOCAL   L1
        CMP     REG,0
        JGE     L1
        NEG     REG
L1:
ENDM

        PUBLIC  MBOX2
MBOX2   PROC    NEAR

        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_X1
        MOV     X1,AX
        MOV     AX,[BP+4].ARG_Y1
        MOV     Y1,AX
        MOV     AX,[BP+4].ARG_X2
        MOV     X2,AX
        MOV     AX,[BP+4].ARG_Y2
        MOV     Y2,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM
        CALL    TEXTW
        CALL    WRITE
        CALL    SET_DX_AND_DY
        CALL    CUL_EAD_AND_DAD
        CALL    CSRW
        CALL    SET_DRAWING_PRAMETER
        CALL    VECTW
        CALL    SET_GC
        CALL    VECTE

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET

TEXTW:
        MOV     COMMAND,78H
        CALL    OUTC
```

```
        MOV     WRDPAR,0FFFFH
        CALL    OUTWP
        RET

WRITE:
        MOV     COMMAND,20H
        CALL    OUTC
        RET

SET_DX_AND_DY:
        MOV     AX,Y1
        CMP     AX,Y2
        JGE     SET1

        MOV     AX,X1
        XCHG    AX,X2
        MOV     X1,AX
        MOV     AX,Y1
        XCHG    AX,Y2
        MOV     Y1,AX
SET1:
        MOV     AX,X2
        SUB     AX,X1
        MOV     DELTA_X,AX
        MOV     AX,Y2
        SUB     AX,Y1
        MOV     DELTA_Y,AX
        RET

SET_DRAWING_PRAMETER:
        MOV     AX,X1
        CMP     AX,X2
        JGE     I1
; 長辺
        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        ADD     AX,BX
        SHR     AX,1
        MOV     A,AX

        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        SUB     AX,BX
        ABS     AX
        SHR     AX,1
```

```
        MOV     B,AX
        JMP     I2
I1:
        JE      I3
; 短辺
        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        ADD     AX,BX
        SHR     AX,1
        MOV     B,AX

        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        SUB     AX,BX
        ABS     AX
        SHR     AX,1
        MOV     A,AX
        JMP     I2
I3:
; 正方形
        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        ADD     AX,BX
        SHR     AX,1
        MOV     A,AX
        MOV     B,AX
I2:

        MOV     DC_REG,3
        MOV     AX,A
        MOV     D_REG,AX
        MOV     AX,B
        MOV     D2_REG,AX
        MOV     D1_REG,-1
        MOV     AX,A
        MOV     DM_REG,AX
        RET

CUL_EAD_AND_DAD:
        MOV     BX,4000H

        MOV     AX,X1
        SHR     AX,1
```

```
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1

        ADD     BX,AX

        MOV     AX,Y1
        MOV     CX,AX
        SHL     CX,1
        SHL     CX,1
        ADD     AX,CX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1

        ADD     BX,AX
        MOV     EAD,BX
        MOV     AX,X1
        AND     AL,15
        MOV     DAD,AL
        RET

CSRW:
        MOV     COMMAND,49H
        CALL    OUTC
        MOV     AX,EAD
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AL,DAD
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1

        MOV     PARMTR,AL
        CALL    OUTP
        RET
VECTW:
        MOV     COMMAND,4CH
        CALL    OUTC
        MOV     PARMTR,43H          ; DIR = 3
        CALL    OUTP
        MOV     AX,DC_REG
        OR      AH,64               ; DGD BIT ON
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,D_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
```

```
        MOV     AX,D2_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,D1_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,DM_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        RET

VECTE:
        MOV     COMMAND,6CH
        CALL    OUTC
        RET

SET_GC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL
        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0
        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
```

```
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL
        RET
;
; コマンド送出ルーチン
;
OUTC:
OUTCLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTCLP
        MOV     AL,COMMAND
        OUT     COUTPT,AL
        RET
;
; パラメータ送出ルーチン
;
OUTP:
OUTPLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTPLP
        MOV     AL,PARMTR
        OUT     POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
OUTWP:
        MOV     AX,WRDPAR
        MOV     PARMTR,AL
        CALL    OUTP
        MOV     PARMTR,AH
        CALL    OUTP
        RET
MBOX2   ENDP
_TEXT   ENDS
        END
```

6 ボックスフィル

ボックスフィルを行うアセンブリ言語定義の関数mbox_ fill( )を示します。

```
void mbox_fill(x1,y1,x2,y2,color)
int x1;        /* X座標1  */
int y1;        /* Y座標1  */
int x2;        /* X座標2  */
int y2;        /* Y座標2  */
int color;     /* 描画色(0≦color≦15) */
```

mbox_ fill( )は，リスト6のようになります。

◉リスト6　"mboxfill.c"

```
;
;  ボックスフィル(アセンブリ言語定義)
;
;       void mboxfill(x1,y1,x2,y2,color);
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;
;
;
; スタック上の引数を表わす
; 構造体
;
ARGS     STRUC
ARG_X1   DW        ?
ARG_Y1   DW        ?
ARG_X2   DW        ?
ARG_Y2   DW        ?
ARG_COLOR          DW        ?
ARGS     ENDS
```

```
;
; データセグメント
;
DGROUP   GROUP    _DATA
_DATA    SEGMENT  WORD PUBLIC 'DATA'
         ASSUME   DS:DGROUP
         PUBLIC   FILL_PATTERN
         PUBLIC   DC_REG
         PUBLIC   D_REG
FILL_PATTERN      DB      0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH

X1       DW       0
Y1       DW       0
X2       DW       0
Y2       DW       0

DELTA_X  DW       0
DELTA_Y  DW       0
COLOR    DW       0

DC_REG   DW       0
D_REG    DW       0

EAD      DW       0
DAD      DB       0
COMMAND  DB       0
PARMTR   DB       0
WRDPAR   DW       0

TILE0    DB       0
TILE1    DB       0
TILE2    DB       0
TILE3    DB       0
_DATA    ENDS

;
; プログラムセグメント
;
_TEXT    SEGMENT  BYTE PUBLIC 'CODE'
         ASSUME   CS:_TEXT

COUTPT   EQU      0A2H
POUTPT   EQU      0A0H
STATPT   EQU      0A0H

ABS      MACRO    REG
         LOCAL    L1
         CMP      REG,0
```

```
        JGE     L1
        NEG     REG
L1:
ENDM

        PUBLIC  MBOX_FILL

MBOX_FILL       PROC    NEAR

        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_X1
        MOV     X1,AX
        MOV     AX,[BP+4].ARG_Y1
        MOV     Y1,AX
        MOV     AX,[BP+4].ARG_X2
        MOV     X2,AX
        MOV     AX,[BP+4].ARG_Y2
        MOV     Y2,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM
        CALL    TEXTW
        CALL    WRITE
        CALL    SET_DX_AND_DY
        CALL    CUL_EAD_AND_DAD
        CALL    CSRW
        CALL    SET_DRAWING_PRAMETER
        CALL    VECTW
        CALL    SET_GC
        CALL    TEXTE

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET

TEXTW:
        MOV     COMMAND,78H
        CALL    OUTC

        MOV     SI,0
```

```
        MOV     CX,8
T1:
        MOV     AL,FILL_PATTERN[SI]
        INC     SI
        MOV     PARMTR,AL
        CALL    OUTP
        LOOP    T1
        RET

WRITE:
        MOV     COMMAND,23H
        CALL    OUTC
        RET

SET_DX_AND_DY:
        MOV     AX,X2
        SUB     AX,X1
        ABS     AX
        MOV     DELTA_X,AX

        MOV     AX,Y2
        SUB     AX,Y1
        ABS     AX
        MOV     DELTA_Y,AX

        MOV     AX,Y2
        CMP     AX,Y1
        JLE     I1
        MOV     AX,Y1
        XCHG    AX,Y2
        MOV     Y1,AX
I1:
        MOV     AX,X2
        CMP     AX,X1
        JGE     I2
        MOV     AX,X1
        XCHG    AX,X2
        MOV     X1,AX
I2:
        RET

SET_DRAWING_PRAMETER:
        MOV     AX,DELTA_Y
        MOV     DC_REG,AX
        MOV     AX,DELTA_X
        MOV     D_REG,AX
        RET
```

```
CUL_EAD_AND_DAD:
        MOV     BX,4000H

        MOV     AX,X1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1

        ADD     BX,AX

        MOV     AX,Y1
        MOV     CX,AX
        SHL     CX,1
        SHL     CX,1
        ADD     AX,CX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1

        ADD     BX,AX
        MOV     EAD,BX
        MOV     AX,X1
        AND     AL,15
        MOV     DAD,AL
        RET
CSRW:
        MOV     COMMAND,49H
        CALL    OUTC
        MOV     AX,EAD
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AL,DAD
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1

        MOV     PARMTR,AL
        CALL    OUTP
        RET
VECTW:
        MOV     COMMAND,4CH
        CALL    OUTC
        MOV     PARMTR,12H          ; DIR = 2
        CALL    OUTP
        MOV     AX,DC_REG
        OR      AH,64               ; DGD BIT ON
```

```
        MOV     WRDPAR,AX
        CALL    OUTWP
        MOV     AX,D_REG
        MOV     WRDPAR,AX
        CALL    OUTWP
        RET

TEXTE:
        MOV     COMMAND,68H
        CALL    OUTC
        RET

SET_GC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0

        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
```

```
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL
        RET

;
; コマンド送出ルーチン
;
OUTC:
OUTCLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTCLP
        MOV     AL,COMMAND
        OUT     COUTPT,AL
        RET

;
; パラメータ送出ルーチン
;
OUTP:
OUTPLP: IN      AL,STATPT
        AND     AL,2
        JNZ     OUTPLP

        MOV     AL,PARMTR
        OUT     POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
OUTWP:
        MOV     AX,WRDPAR
        MOV     PARMTR,AL
        CALL    OUTP
        MOV     PARMTR,AH
        CALL    OUTP
        RET

MBOX_FILL       ENDP
_TEXT   ENDS
        END
```

7 扇形描画関数

扇形を描画するアセンブリ言語定義の関数marc2( )を示します。

```
void marc2(x,y,r,color,start,end)
 int x;          /* 中心X座標 */
 int y;          /* 中心Y座標 */
 int r;          /* 半径   */
 int color;      /* 描画色(0≦color≦15)  */
 int start;      /* 開始角度（度の１０倍）*/
                 /*  0≦ start ≦3599    */
 int end;        /* 終了角度（度の１０倍）*/
                 /*  0≦ end   ≦3599    */
```

関数marc2( )の定義は，リスト７のようになります。

◉リスト７

```
;
;
; 扇形描画関数(アセンブリ言語定義)
;
;         void marc2(x,y,r,color,start,end);
;
;
;         スモールモデル
;         GDC  5MHz
;         グラフィックチャージャ使用
;
;         SAVE "MARC2.ASM"
;

;
; スタック上の引数を表わす
; 構造体
;
ARGS      STRUC
ARG_X     DW     ?
ARG_Y     DW     ?
ARG_R     DW     ?
```

```
ARG_COLOR        DW      ?
ARG_START        DW      ?
ARG_END          DW      ?
ARGS    ENDS

;
; データセグメント
;
DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP

X_C     DW      0
Y_C     DW      0
RADIUS  DW      0
ALPHA   DW      0
BETA    DW      0
DIR     DW      0

DC_REG  DW      0
D_REG   DW      0
D1_REG  DW      0
D2_REG  DW      0
DM_REG  DW      0

EAD     DW      0
DAD     DB      0

XCRD    DW      0
YCRD    DW      0

I       DW      0
J       DW      0
N       DW      0

A       DW      0
B       DW      0
AA      DW      0
BB      DW      0

IA      DW      0
IB      DW      0
II      DW      0

START_ANGLE     DW      0
END_ANGLE       DW      0

COLOR   DW      0
TILE0   DB      0
```

```
TILE1     DB      0
TILE2     DB      0
TILE3     DB      0

LINE_X1 DW      0
LINE_Y1 DW      0
LINE_X2 DW      0
LINE_Y2 DW      0

XX1     DW      0
YY1     DW      0
XX2     DW      0
YY2     DW      0
COLOR0  DW      0

X1      DW      0
Y1      DW      0
X2      DW      0
Y2      DW      0

DELTA_X DW      0
DELTA_Y DW      0

SINTABLE        DW        0,   1,   3,   5,   6,   8,  10,  12,  13,  15
                DW       17,  19,  20,  22,  24,  26,  27,  29,  31,  33
                DW       34,  36,  38,  40,  41,  43,  45,  47,  48,  50
                DW       52,  54,  55,  57,  59,  61,  62,  64,  66,  68
                DW       69,  71,  73,  74,  76,  78,  80,  81,  83,  85
                DW       87,  88,  90,  92,  94,  95,  97,  99, 101, 102
                DW      104, 106, 107, 109, 111, 113, 114, 116, 118, 120
                DW      121, 123, 125, 127, 128, 130, 132, 133, 135, 137
                DW      139, 140, 142, 144, 146, 147, 149, 151, 152, 154
                DW      156, 158, 159, 161, 163, 165, 166, 168, 170, 171
                DW      173, 175, 177, 178, 180, 182, 183, 185, 187, 189
                DW      190, 192, 194, 195, 197, 199, 201, 202, 204, 206
                DW      207, 209, 211, 213, 214, 216, 218, 219, 221, 223
                DW      224, 226, 228, 230, 231, 233, 235, 236, 238, 240
                DW      241, 243, 245, 246, 248, 250, 252, 253, 255, 257
                DW      258, 260, 262, 263, 265, 267, 268, 270, 272, 273
                DW      275, 277, 278, 280, 282, 284, 285, 287, 289, 290
                DW      292, 294, 295, 297, 299, 300, 302, 304, 305, 307
                DW      309, 310, 312, 313, 315, 317, 318, 320, 322, 323
                DW      325, 327, 328, 330, 332, 333, 335, 337, 338, 340
                DW      342, 343, 345, 346, 348, 350, 351, 353, 355, 356
                DW      358, 359, 361, 363, 364, 366, 368, 369, 371, 372
                DW      374, 376, 377, 379, 381, 382, 384, 385, 387, 389
                DW      390, 392, 393, 395, 397, 398, 400, 401, 403, 405
                DW      406, 408, 409, 411, 413, 414, 416, 417, 419, 421
                DW      422, 424, 425, 427, 428, 430, 432, 433, 435, 436
                DW      438, 439, 441, 443, 444, 446, 447, 449, 450, 452
                DW      453, 455, 457, 458, 460, 461, 463, 464, 466, 467
                DW      469, 471, 472, 474, 475, 477, 478, 480, 481, 483
                DW      484, 486, 487, 489, 490, 492, 493, 495, 496, 498
```

```
        DW      500, 501, 503, 504, 506, 507, 509, 510, 512, 513
        DW      515, 516, 518, 519, 521, 522, 523, 525, 526, 528
        DW      529, 531, 532, 534, 535, 537, 538, 540, 541, 543
        DW      544, 546, 547, 549, 550, 551, 553, 554, 556, 557
        DW      559, 560, 562, 563, 564, 566, 567, 569, 570, 572
        DW      573, 575, 576, 577, 579, 580, 582, 583, 584, 586
        DW      587, 589, 590, 592, 593, 594, 596, 597, 599, 600
        DW      601, 603, 604, 605, 607, 608, 610, 611, 612, 614
        DW      615, 617, 618, 619, 621, 622, 623, 625, 626, 627
        DW      629, 630, 632, 633, 634, 636, 637, 638, 640, 641
        DW      642, 644, 645, 646, 648, 649, 650, 652, 653, 654
        DW      656, 657, 658, 660, 661, 662, 663, 665, 666, 667
        DW      669, 670, 671, 673, 674, 675, 676, 678, 679, 680
        DW      681, 683, 684, 685, 687, 688, 689, 690, 692, 693
        DW      694, 695, 697, 698, 699, 700, 702, 703, 704, 705
        DW      707, 708, 709, 710, 712, 713, 714, 715, 716, 718
        DW      719, 720, 721, 722, 724, 725, 726, 727, 728, 730
        DW      731, 732, 733, 734, 736, 737, 738, 739, 740, 741
        DW      743, 744, 745, 746, 747, 748, 750, 751, 752, 753
        DW      754, 755, 756, 758, 759, 760, 761, 762, 763, 764
        DW      766, 767, 768, 769, 770, 771, 772, 773, 774, 776
        DW      777, 778, 779, 780, 781, 782, 783, 784, 785, 786
        DW      788, 789, 790, 791, 792, 793, 794, 795, 796, 797
        DW      798, 799, 800, 801, 802, 803, 804, 805, 806, 807
        DW      809, 810, 811, 812, 813, 814, 815, 816, 817, 818
        DW      819, 820, 821, 822, 823, 824, 825, 826, 827, 828
        DW      829, 830, 830, 831, 832, 833, 834, 835, 836, 837
        DW      838, 839, 840, 841, 842, 843, 844, 845, 846, 847
        DW      848, 848, 849, 850, 851, 852, 853, 854, 855, 856
        DW      857, 858, 858, 859, 860, 861, 862, 863, 864, 865
        DW      866, 866, 867, 868, 869, 870, 871, 872, 872, 873
        DW      874, 875, 876, 877, 877, 878, 879, 880, 881, 882
        DW      882, 883, 884, 885, 886, 887, 887, 888, 889, 890
        DW      891, 891, 892, 893, 894, 894, 895, 896, 897, 898
        DW      898, 899, 900, 901, 901, 902, 903, 904, 904, 905
        DW      906, 907, 907, 908, 909, 909, 910, 911, 912, 912
        DW      913, 914, 914, 915, 916, 917, 917, 918, 919, 919
        DW      920, 921, 921, 922, 923, 923, 924, 925, 925, 926
        DW      927, 927, 928, 929, 929, 930, 931, 931, 932, 932
        DW      933, 934, 934, 935, 936, 936, 937, 937, 938, 939
        DW      939, 940, 940, 941, 942, 942, 943, 943, 944, 944
        DW      945, 946, 946, 947, 947, 948, 948, 949, 949, 950
        DW      951, 951, 952, 952, 953, 953, 954, 954, 955, 955
        DW      956, 956, 957, 957, 958, 958, 959, 959, 960, 960
        DW      961, 961, 962, 962, 963, 963, 964, 964, 965, 965
        DW      965, 966, 966, 967, 967, 968, 968, 969, 969, 969
        DW      970, 970, 971, 971, 971, 972, 972, 973, 973, 973
        DW      974, 974, 975, 975, 975, 976, 976, 977, 977, 977
        DW      978, 978, 978, 979, 979, 979, 980, 980, 980, 981
        DW      981, 981, 982, 982, 982, 983, 983, 983, 984, 984
        DW      984, 985, 985, 985, 985, 986, 986, 986, 987, 987
        DW      987, 987, 988, 988, 988, 989, 989, 989, 989, 990
        DW      990, 990, 990, 990, 991, 991, 991, 991, 992, 992
```

```
        DW      992, 992, 992, 993, 993, 993, 993, 993, 994, 994
        DW      994, 994, 994, 995, 995, 995, 995, 995, 995, 996
        DW      996, 996, 996, 996, 996, 996, 997, 997, 997, 997
        DW      997, 997, 997, 997, 998, 998, 998, 998, 998, 998
        DW      998, 998, 998, 998, 998, 999, 999, 999, 999, 999
        DW      999, 999, 999, 999, 999, 999, 999, 999, 999, 999
        DW      999, 999, 999, 999, 999, 999, 999, 999, 999, 999
        DW     1000
_DATA   ENDS

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT
;
; 定数
;
COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
STATPT  EQU     0A0H

        PUBLIC  MARC2

MARC2   PROC    NEAR
        PUSH    BP
        MOV     BP, SP

        MOV     AX, [BP+4].ARG_X
        MOV     X_C, AX
        MOV     AX, [BP+4].ARG_Y
        MOV     Y_C, AX
        MOV     AX, [BP+4].ARG_R
        MOV     RADIUS, AX
        MOV     AX, [BP+4].ARG_COLOR
        MOV     COLOR, AX
        MOV     AX, [BP+4].ARG_START
        MOV     START_ANGLE, AX
        MOV     AX, [BP+4].ARG_END
        MOV     END_ANGLE, AX

        IRP     REG, <BX, CX, DX, SI, DI, DS, ES>
                PUSH    REG
        ENDM

        MOV     AX, START_ANGLE
        CALL    ICOS
        IMUL    RADIUS
        MOV     BX, 1000
        IDIV    BX
        ADD     AX, X_C
        MOV     LINE_X1, AX
```

```
        MOV     AX, START_ANGLE
        CALL    ISIN
        IMUL    RADIUS
        MOV     BX, 1000
        IDIV    BX
        ADD     AX, Y_C
        MOV     LINE_Y1, AX
        MOV     AX, END_ANGLE
        CALL    ICOS
        IMUL    RADIUS
        MOV     BX, 1000
        IDIV    BX
        ADD     AX, X_C
        MOV     LINE_X2, AX

        MOV     AX, END_ANGLE
        CALL    ISIN
        IMUL    RADIUS
        MOV     BX, 1000
        IDIV    BX
        ADD     AX, Y_C
        MOV     LINE_Y2, AX
        CALL    ARC

        IRP     REG, <ES, DS, DI, SI, DX, CX, BX>
        POP     REG
        ENDM

        MOV     SP, BP
        POP     BP
        RET

ARC00:
        MOV     ALPHA, AX
        MOV     BETA, BX
        MOV     DIR, CX

        MOV     AX, BETA
        CALL    ISIN
        IMUL    RADIUS
        MOV     BX, 1000
        IDIV    BX
        OR      AH, 64          ; DGD bit ON
        MOV     DC_REG, AX

        MOV     AX, RADIUS
        DEC     AX
        MOV     D_REG, AX
```

```
        MOV     AX,D_REG
        SHL     AX,1
        MOV     D2_REG,AX

        MOV     D1_REG,-1

        MOV     AX,ALPHA
        CALL    ISIN
        IMUL    RADIUS
        MOV     BX,1000
        IDIV    BX
        MOV     DM_REG,AX

        CMP     DIR,7
        JE      L01
        CMP     DIR,4
        JE      L01
        JMP     L1
L01:
        MOV     AX,X_C
        ADD     AX,RADIUS
        MOV     XCRD,AX

        MOV     AX,Y_C
        MOV     YCRD,AX

L1:
        CMP     DIR,0
        JE      L02
        CMP     DIR,3
        JE      L02
        JMP     L2
L02:
        MOV     AX,X_C
        SUB     AX,RADIUS
        MOV     XCRD,AX

        MOV     AX,Y_C
        MOV     YCRD,AX

L2:
        CMP     DIR,1
        JE      L03
        CMP     DIR,6
        JE      L03
        JMP     L3
L03:
        MOV     AX,X_C
        MOV     XCRD,AX
```

```
        MOV     AX,Y_C
        SUB     AX,RADIUS
        MOV     YCRD,AX
L3:
        CMP     DIR,2
        JE      L04
        CMP     DIR,5
        JE      L04
        JMP     L4
L04:
        MOV     AX,X_C
        MOV     XCRD,AX
        MOV     AX,Y_C
        ADD     AX,RADIUS
        MOV     YCRD,AX
L4:
        MOV     AX,YCRD
        MOV     BX,AX
        SHL     BX,1
        SHL     BX,1
        ADD     AX,BX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1
        MOV     CX,AX
        MOV     AX,XCRD
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        ADD     AX,CX
        MOV     EAD,AX

        MOV     AX,XCRD
        AND     AX,000FH
        MOV     DAD,AL

CSRW1:
        MOV     AX,49H
        CALL    SENDC
        MOV     AX,EAD
        CALL    SENDWP
```

```
        MOV     AL,DAD
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1
        CALL    SENDP

        MOV     AX,4CH
        CALL    SENDC

        MOV     AX,20H
        ADD     AX,DIR
        CALL    SENDP

        MOV     AX,DC_REG
        CALL    SENDWP

        MOV     AX,D_REG
        CALL    SENDWP

        MOV     AX,D2_REG
        CALL    SENDWP

        MOV     AX,D1_REG
        CALL    SENDWP

        MOV     AX,DM_REG
        CALL    SENDWP

        MOV     AX,6CH
        CALL    SENDC

        RET
ARC0_INTERNAL:
        MOV     A,AX
        MOV     B,BX
        MOV     II,CX
        MOV     AX,II
        MOV     BX,AX
        SHL     BX,1
        ADD     AX,BX
        ADD     AX,7
        AND     AX,7
        MOV     DIR,AX

        MOV     AX,II
        TEST    AX,1
```

```
        JNZ     I1
        MOV     AX,A
        MOV     BX,B
        MOV     CX,DIR
        CALL    ARC00
        JMP     I2

I1:
        MOV     AX,450
        SUB     AX,B
        MOV     BX,450
        SUB     BX,A
        MOV     CX,DIR
        CALL    ARC00
I2:
        RET


ARC0:
        MOV     AX,78H
        CALL    SENDC

        MOV     AX,0FFFFH
        CALL    SENDWP

        MOV     AX,START_ANGLE
        CWD
        MOV     BX,450
        IDIV    BX
        MOV     IA,AX
        MOV     AA,DX

        MOV     AX,END_ANGLE
        CWD
        MOV     BX,450
        IDIV    BX
        MOV     IB,AX
        MOV     BB,DX

        MOV     AX,IB
        SUB     AX,IA
        ADD     AX,7
        CWD
        MOV     BX,8
        IDIV    BX
        MOV     N,DX
        MOV     AX,IA
```

```
        CMP     AX,IB
        JNE     A1

        MOV     AX,START_ANGLE
        CMP     AX,END_ANGLE
        JG      A1

        MOV     AX,AA
        MOV     BX,BB
        MOV     CX,IA
        CALL    ARC0_INTERNAL
        JMP     A2
A1:
        MOV     AX,AA
        MOV     BX,450
        MOV     CX,IA
        CALL    ARC0_INTERNAL

        MOV     J,1
FOR1:
        MOV     AX,J
        CMP     AX,N
        JG      NEXT1

        MOV     AX,IA
        ADD     AX,J
        AND     AX,7
        MOV     I,AX

        MOV     AX,0
        MOV     BX,450
        MOV     CX,I
        CALL    ARC0_INTERNAL
        INC     J
        JMP     FOR1
NEXT1:
        MOV     AX,0
        MOV     BX,BB
        MOV     CX,IB
        CALL    ARC0_INTERNAL
A2:
        RET

ARC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
```

```
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0
        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL

; WRITE COMMAND
        MOV     AX,023H
        CALL    SENDC
        CALL    ARC0

        MOV     AX,X_C
        MOV     BX,Y_C
        MOV     CX,LINE_X1
        MOV     DX,LINE_Y1
        CALL    MLINELIB

        MOV     AX,X_C
        MOV     BX,Y_C
        MOV     CX,LINE_X2
        MOV     DX,LINE_Y2
```

```
        CALL     MLINELIB
        RET
;
; コマンド送出ルーチン
;
SENDC:
        MOV      BX,AX
SENDCLP:        IN       AL,STATPT
        AND      AL,2
        JNZ      SENDCLP
        MOV      AX,BX
        OUT      COUTPT,AL
        RET

;
; パラメータ送出ルーチン
;
SENDP:
        MOV      BX,AX
SENDPLP:        IN       AL,STATPT
        AND      AL,2
        JNZ      SENDPLP

        MOV      AX,BX
        OUT      POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
SENDWP:
        MOV      CX,AX
        CALL     SENDP
        MOV      AL,CH
        CALL     SENDP
        RET

ISIN:
        CWD
        MOV      BX,3600
        IDIV     BX
        MOV      DI,0               ; SIGN
        CMP      DX,0
        JGE      S1
        MOV      DI,0FFFFH          ; MINUS
        NEG      DX
S1:
        CMP      DX,900
        JG       S2
```

```
        MOV     SI,DX
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        JMP     SINEND
S2:
        CMP     DX,1800
        JG      S3
        MOV     SI,1800
        SUB     SI,DX
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        JMP     SINEND
S3:
        CMP     DX,2700
        JG      S4
        MOV     SI,DX
        SUB     SI,1800
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        NEG     AX
        JMP     SINEND
S4:
        MOV     SI,3600
        SUB     SI,DX
        SHL     SI,1
        MOV     AX,SINTABLE[SI]
        NEG     AX
SINEND:
        CMP     DI,0
        JNE     S5
        RET
S5:
        NEG     AX
        RET

ICOS:
        ADD     AX,900
        CALL    ISIN
        RET

MARC2   ENDP

MLINELIB        PROC NEAR
        MOV     X1,AX
        MOV     Y1,BX
        MOV     X2,CX
        MOV     Y2,DX

INIT:                           ; LET DELTA_X=X2-X1>=0
```

```
        MOV     AX,X2
        CMP     AX,X1
        JGE     INIT1              ; IF DELTA_X>=0 THEN INIT1

        MOV     BX,X1              ; SWAP X1,X2
        MOV     X1,AX
        MOV     X2,BX
        MOV     AX,Y1              ; SWAP Y1,Y2
        MOV     BX,Y2
        MOV     Y2,AX
        MOV     Y1,BX
INIT1:  MOV     AX,X1
        CMP     AX,X2
        JNE     INIT2              ; IF X1<>X2 THEN INIT2

        MOV     AX,Y2
        CMP     AX,Y1
        JGE     INIT3              ; IF Y2>=Y1 THEN INIT3

        MOV     BX,Y1              ; SWAP Y1,Y2
        MOV     Y1,AX
        MOV     Y2,BX

INIT2:
INIT3:

CULADR:
        MOV     CX,4000H           ; LET V-RAM START ADRESS=4000H

        MOV     AX,X1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1               ; AX=INT(X1/16)

        ADD     CX,AX

        MOV     AX,Y1
        MOV     BX,AX
        SHL     BX,1
        SHL     BX,1
        ADD     AX,BX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1               ; AX=40*Y1

        ADD     CX,AX
        MOV     EAD,CX
```

```
        MOV     AX,X1
        AND     AX,0FH              ; AX=X1 MOD 16
        MOV     DAD,AL
;
CSRW2:
        MOV     AX,49H              ; CSRW COMMAND=49H
        CALL    SENDC

        MOV     AX,EAD
        CALL    SENDWP

        MOV     AL,DAD
        MOV     CL,4
        SHL     AL,CL               ; SHIFT DAD 4 TIMES
        CALL    SENDP
;
CULDIR:
        MOV     AX,X2
        SUB     AX,X1
        MOV     DELTA_X,AX                  ; DELTA_X=X2-X1

        MOV     AX,Y2
        SUB     AX,Y1
        MOV     DELTA_Y,AX                  ; DELTA_Y=Y2-Y1

        MOV     AX,DELTA_Y
        CMP     AX,0
        JLE     DIR1                ; IF DELTA_Y<=0 THEN DIR1

        MOV     BX,DELTA_X
        CMP     AX,BX
        JLE     DIR2

        XOR     AX,AX
        MOV     DIR,AX              ; DIR = 0

        MOV     AX,DELTA_X
        MOV     BX,DELTA_Y
        MOV     DELTA_Y,AX
        MOV     DELTA_X,BX                  ; SWAP DELTA_X,DELTA_Y

        JMP     DIR3

DIR2:   MOV     AX,1
        MOV     DIR,AX              ; DIR=1
        JMP     DIR3

DIR1:   MOV     BX,DELTA_Y
        NEG     BX
```

```
        MOV     DELTA_Y,BX                      ; DELTA_Y=ABS(DELTA_Y)

        MOV     AX,DELTA_X
        CMP     BX,AX
        JG      DIR4                  ; IF DELTA_Y>DELTA_X THEN DIR4

        MOV     AX,2
        MOV     DIR,AX                ; DIR=2
        JMP     DIR3

DIR4:   MOV     AX,3
        MOV     DIR,AX

        MOV     AX,DELTA_X
        MOV     BX,DELTA_Y
        MOV     DELTA_Y,AX
        MOV     DELTA_X,BX                      ; SWAP DELTA_X,DELTA_Y

DIR3:
;
CULPAR:
        MOV     AX,DELTA_X
        MOV     DC_REG,AX             ; LET DC_REG=DELTA_X

        MOV     AX,DELTA_Y
        SAL     AX,1
        SAL     AX,1
        SUB     AX,DELTA_X
        MOV     D_REG,AX              ; LET D_REG=2*DELTA_Y-DELTA_X

        MOV     AX,DELTA_Y
        SUB     AX,DELTA_X
        SAL     AX,1
        SAL     AX,1
        MOV     D2_REG,AX             ; LET D2_REG=2*DELTA_Y-2*DELTA_X

        MOV     AX,DELTA_Y
        SAL     AX,1
        SAL     AX,1
        MOV     D1_REG,AX             ; LET D1_REG=2*DELTA_Y
;
VECTW:
        MOV     AL,4CH                ; VECTW COMMAND=4CH
        CALL    SENDC

        MOV     AX,DIR
        OR      AL,8                  ; DECLEAR TO DRAW 'LINE'

        CALL    SENDP
```

```
        MOV     AX,DC_REG
        OR      AH,64               ; DGD BIT ON
        CALL    SENDWP

        MOV     AX,D_REG
        CALL    SENDWP

        MOV     AX,D2_REG
        CALL    SENDWP

        MOV     AX,D1_REG
        CALL    SENDWP
VECTE:
        MOV     AL,6CH              ; VECTE COMMAND=6CH
        CALL    SENDC

        RET
;
;       END OF LINE ROUTINE

MLINELIB        ENDP
_TEXT   ENDS
        END
```

8 傾斜四辺形塗りつぶし関数

アセンブリ言語で定義した傾斜四辺形を塗りつぶす関数mbox_fill2( )を示します。

```
void mbox_fill2(x1,y1,x2,y2,color)
int x1;        /* X座標1  */
int y1;        /* Y座標1  */
int x2;        /* X座標2  */
int y2;        /* Y座標2  */
int color;     /* 描画色(0≦color≦15) */
```

関数mbox_fill2( )の定義は，リスト8のようになります。

◉リスト8

```
;
;   傾斜四辺形塗りつぶし関数(アセンブリ言語定義)
;
;       void mbox_fill2(x1,y1,x2,y2,color);
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;

;
; スタック上の引数を表わす
; 構造体
;
ARGS    STRUC
ARG_X1  DW      ?
ARG_Y1  DW      ?
ARG_X2  DW      ?
ARG_Y2  DW      ?
ARG_COLOR       DW      ?
ARGS    ENDS
```

```
;
; データセグメント
;
DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP
        PUBLIC  FILL_PATTERN

FILL_PATTERN    DB      0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH,0FFH

A       DW      0
B       DW      0

X1      DW      0
Y1      DW      0
X2      DW      0
Y2      DW      0

DELTA_X DW      0
DELTA_Y DW      0
COLOR   DW      0

DC_REG  DW      0
D_REG   DW      0

EAD     DW      0
DAD     DB      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0
_DATA   ENDS

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
STATPT  EQU     0A0H

ABS     MACRO   REG
        LOCAL   L1
        CMP     REG,0
        JGE     L1
```

```
        NEG     REG
L1:
ENDM

        PUBLIC  MBOX_FILL2

MBOX_FILL2      PROC    NEAR

        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_X1
        MOV     X1,AX
        MOV     AX,[BP+4].ARG_Y1
        MOV     Y1,AX
        MOV     AX,[BP+4].ARG_X2
        MOV     X2,AX
        MOV     AX,[BP+4].ARG_Y2
        MOV     Y2,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM
        CALL    TEXTW
        CALL    WRITE
        CALL    SET_DX_AND_DY
        CALL    CUL_EAD_AND_DAD
        CALL    CSRW
        CALL    SET_DRAWING_PRAMETER
        CALL    VECTW
        CALL    SET_GC
        CALL    TEXTE

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET

TEXTW:
        MOV     AL,78H
        CALL    SENDC

        MOV     SI,0
        MOV     CX,8
```

```
T1:
        MOV     AL,FILL_PATTERN[SI]
        INC     SI
        CALL    SENDP
        LOOP    T1
        RET

WRITE:
        MOV     AL,23H
        CALL    SENDC
        RET

SET_DX_AND_DY:
        MOV     AX,Y1
        CMP     AX,Y2
        JGE     L1

        MOV     AX,X1
        XCHG    AX,X2
        MOV     X1,AX

        MOV     AX,Y1
        XCHG    AX,Y2
        MOV     Y1,AX
L1:
        MOV     AX,X2
        SUB     AX,X1
        MOV     DELTA_X,AX

        MOV     AX,Y2
        SUB     AX,Y1
        MOV     DELTA_Y,AX

        MOV     AX,X1
        CMP     AX,X2
        JGE     L3

        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        ADD     AX,BX
        SHR     AX,1
        MOV     A,AX

        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
```

```
        ABS     BX
        SUB     AX,BX
        ABS     AX
        SHR     AX,1
        MOV     B,AX
L3:
        MOV     AX,X1
        CMP     AX,X2
        JLE     L4

        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        ADD     AX,BX
        SHR     AX,1
        MOV     B,AX

        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        SUB     AX,BX
        ABS     AX
        SHR     AX,1
        MOV     A,AX
L4:
        MOV     AX,X1
        CMP     AX,X2
        JNE     L5

        MOV     AX,DELTA_X
        ABS     AX
        MOV     BX,DELTA_Y
        ABS     BX
        ADD     AX,BX
        SHR     AX,1
        MOV     A,AX
        MOV     B,AX
L5:
        RET

SET_DRAWING_PRAMETER:
        MOV     AX,B
        MOV     DC_REG,AX
        MOV     AX,A
        MOV     D_REG,AX
        RET
```

```
CUL_EAD_AND_DAD:
        MOV     BX, X1
        SHR     BX, 1
        SHR     BX, 1
        SHR     BX, 1
        SHR     BX, 1

        ADD     BX, 4000H

        MOV     AX, Y1
        MOV     CX, AX
        SHL     CX, 1
        SHL     CX, 1
        ADD     AX, CX
        SHL     AX, 1
        SHL     AX, 1
        SHL     AX, 1

        ADD     BX, AX
        MOV     EAD, BX
        MOV     AX, X1
        AND     AL, 15
        MOV     DAD, AL
        RET
CSRW:
        MOV     AL, 49H
        CALL    SENDC

        MOV     AX, EAD
        CALL    SENDWP

        MOV     AL, DAD
        SHL     AL, 1
        SHL     AL, 1
        SHL     AL, 1
        SHL     AL, 1

        CALL    SENDP
        RET
VECTW:
        MOV     AL, 4CH
        CALL    SENDC

        MOV     AL, 13H
        CALL    SENDP                   ; DIR = 3

        MOV     AX, DC_REG
        OR      AH, 64                  ; DGD BIT ON
```

```
        CALL    SENDWP

        MOV     AX,D_REG
        CALL    SENDWP
        RET

TEXTE:
        MOV     AL,68H
        CALL    SENDC
        RET

SET_GC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0
        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
```

```
        OUT     07EH,AL
        RET

;
; コマンド送出ルーチン
;
SENDC:
        MOV     BX,AX
SENDCLP:        IN      AL,STATPT
        AND     AL,2
        JNZ     SENDCLP
        MOV     AX,BX
        OUT     COUTPT,AL
        RET

;
; パラメータ送出ルーチン
;
SENDP:
        MOV     BX,AX
SENDPLP:        IN      AL,STATPT
        AND     AL,2
        JNZ     SENDPLP

        MOV     AX,BX
        OUT     POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
SENDWP:
        MOV     CX,AX
        CALL    SENDP
        MOV     AL,CH
        CALL    SENDP
        RET
MBOX_FILL2      ENDP
_TEXT   ENDS
        END
```

9 グラフィックス文字描画

描画例で解説したグラフィックス文字描画関数 gputc( )を，アセンブリ言語で定義した例をリスト 9 に示します。関数名は mgputc( )で引数などは同じです。GDC5MHz，グラフィックチャージャを使用しています。

●リスト 9

```
;
;  グラフィックス文字描画(アセンブリ言語定義)
;
;
;       void mgputc(ascii_code,x,y,color);
;
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;
;


;
; スタック上の引数を表わす
; 構造体
;
ARGS    STRUC
ARG_ASCII_CODE  DW      ?
ARG_X           DW      ?
ARG_Y           DW      ?
ARG_COLOR       DW      ?
ARGS    ENDS


;
; データセグメント
;
DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP

BUFFER  DB      34 DUP(?)

PATTERN DB      8 DUP(?)

ASCII_CODE      DW      0
X       DW      0
```

```
Y       DW      0
COLOR   DW      0

DC_REG  DW      0

EAD     DW      0
DAD     DB      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0

_DATA   ENDS

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
STATPT  EQU     0A0H

        PUBLIC  MGPUTC

MGPUTC  PROC    NEAR

        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_ASCII_CODE
        MOV     ASCII_CODE,AX
        MOV     AX,[BP+4].ARG_X
        MOV     X,AX
        MOV     AX,[BP+4].ARG_Y
        MOV     Y,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM

        CALL    SET_PATTERN
        CALL    TEXTW
        CALL    WRITE
        CALL    CSRW
```

```
        CALL    VECTW
        CALL    SET_GC
        CALL    TEXTE

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET

SET_PATTERN:
; ASCII_CODE -->
        MOV     AH,14H
        MOV     BX,DS
        MOV     CX,OFFSET BUFFER
        MOV     DX,ASCII_CODE
        INT     18H

        XOR     SI,SI

        MOV     CX,8
S0:
        PUSH    CX
        MOV     AL,BUFFER[SI+2]

        MOV     CX,8        ; ビット列を反転
        XOR     AH,AH
S1:
        SHL     AL,1
        RCR     AH,1
        LOOP    S1
        MOV     PATTERN[SI],AH
        INC     SI
        POP     CX
        LOOP    S0
        RET

TEXTW:
        MOV     AL,78H
        CALL    SENDC

        MOV     SI,0
        MOV     CX,8
T1:
        MOV     AL,PATTERN[SI]
        INC     SI
```

```
	CALL	SENDP
	LOOP	T1
	RET
WRITE:
	MOV	AL,23H
	CALL	SENDC
	RET
CSRW:
	MOV	BX,X
	SHR	BX,1
	SHR	BX,1
	SHR	BX,1
	SHR	BX,1

	ADD	BX,4000H

	MOV	AX,Y
	MOV	CX,AX
	SHL	CX,1
	SHL	CX,1
	ADD	AX,CX
	SHL	AX,1
	SHL	AX,1
	SHL	AX,1

	ADD	BX,AX
	MOV	EAD,BX
	MOV	AX,X
	AND	AL,15
	MOV	DAD,AL

	MOV	AL,49H
	CALL	SENDC

	MOV	AX,EAD
	CALL	SENDWP

	MOV	AL,DAD
	SHL	AL,1
	SHL	AL,1
	SHL	AL,1
	SHL	AL,1
	CALL	SENDP
	RET
VECTW:
	MOV	DC_REG,7
```

```
        MOV     AL,4CH
        CALL    SENDC

        MOV     AL,12H
        CALL    SENDP                   ; DIR = 2

        MOV     AX,DC_REG
        OR      AH,64                   ; DGD BIT ON
        CALL    SENDWP
        RET

TEXTE:
        MOV     AL,68H
        CALL    SENDC
        RET

SET_GC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0

        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
M4:
        MOV     AL,TILE0
```

```
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL
        RET

;
; コマンド送出ルーチン
;
SENDC:
        MOV     BX,AX
SENDCLP:        IN      AL,STATPT
        AND     AL,2
        JNZ     SENDCLP
        MOV     AX,BX
        OUT     COUTPT,AL
        RET

;
; パラメータ送出ルーチン
;
SENDP:
        MOV     BX,AX
SENDPLP:        IN      AL,STATPT
        AND     AL,2
        JNZ     SENDPLP

        MOV     AX,BX
        OUT     POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
SENDWP:
        MOV     CX,AX
        CALL    SENDP
        MOV     AL,CH
        CALL    SENDP
        RET

MGPUTC  ENDP
_TEXT   ENDS
        END
```

10 グラフィックス文字描画(傾斜ズーム可能)

傾斜/ズーム可能なグラフィックス文字描画関数 gputc2( )を，アセンブリ言語で定義した関数 mgputc2( )をリスト 10 に示します。

◉リスト 10

```
;
;  グラフィックス文字描画
;  傾斜ズーム可能
;       void mgputc2(pattern,x,y,color,zoom,dir,slant);
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;

;
; スタック上の引数を表わす
; 構造体
;
ARGS    STRUC
ARG_PATTERN     DW      ?
ARG_X           DW      ?
ARG_Y           DW      ?
ARG_COLOR       DW      ?
ARG_ZOOM        DW      ?
ARG_DIR         DW      ?
ARG_SLANT       DW      ?
ARGS    ENDS

;
; データセグメント
;
DGROUP  GROUP   _DATA
_DATA   SEGMENT WORD PUBLIC 'DATA'
        ASSUME  DS:DGROUP

PATTERN_PTR     DW      0

ASCII_CODE      DW      0
X       DW      0
```

```
Y       DW      0
COLOR   DW      0

DC_REG  DW      0

EAD     DW      0
DAD     DB      0

TILE0   DB      0
TILE1   DB      0
TILE2   DB      0
TILE3   DB      0

DIR     DW      0
SLANT   DW      0
ZOOM    DW      0
_DATA   ENDS

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

COUTPT  EQU     0A2H
POUTPT  EQU     0A0H
STATPT  EQU     0A0H
TRUE    EQU     -1
FALSE   EQU     0
        PUBLIC  MGPUTC2

MGPUTC2 PROC    NEAR

        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_PATTERN
        MOV     PATTERN_PTR,AX
        MOV     AX,[BP+4].ARG_X
        MOV     X,AX
        MOV     AX,[BP+4].ARG_Y
        MOV     Y,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        MOV     AX,[BP+4].ARG_ZOOM
        MOV     ZOOM,AX
        MOV     AX,[BP+4].ARG_DIR
        MOV     DIR,AX
```

```
        MOV       AX,[BP+4].ARG_SLANT
        MOV       SLANT,AX

        IRP       REG,<BX,CX,DX,SI,DI,DS,ES>
                  PUSH      REG
        ENDM

        CALL      ZOOMW
        CALL      TEXTW
        CALL      WRITE
        CALL      CSRW
        CALL      VECTW
        CALL      SET_GC
        CALL      TEXTE

        IRP       REG,<ES,DS,DI,SI,DX,CX,BX>
        POP       REG
        ENDM

        MOV       SP,BP
        POP       BP
        RET

ZOOMW:
        MOV       AL,46H
        CALL      SENDC
        MOV       AX,ZOOM
        CALL      SENDP
        RET

TEXTW:
        MOV       AL,78H
        CALL      SENDC

        MOV       SI,0
        MOV       CX,8
        MOV       BX,PATTERN_PTR
 T1:
        MOV       AL,[BX+SI]
        INC       SI

        PUSH      BX
        CALL      SENDP
        POP       BX

        LOOP      T1
        RET
```

```
WRITE:
        MOV     AL,23H
        CALL    SENDC
        RET

CSRW:
        MOV     BX,X
        SHR     BX,1
        SHR     BX,1
        SHR     BX,1
        SHR     BX,1

        ADD     BX,4000H

        MOV     AX,Y
        MOV     CX,AX
        SHL     CX,1
        SHL     CX,1
        ADD     AX,CX
        SHL     AX,1
        SHL     AX,1
        SHL     AX,1

        ADD     BX,AX
        MOV     EAD,BX
        MOV     AX,X
        AND     AL,15
        MOV     DAD,AL

        MOV     AL,49H
        CALL    SENDC

        MOV     AX,EAD
        CALL    SENDWP

        MOV     AL,DAD
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1

        CALL    SENDP
        RET
VECTW:
        MOV     DC_REG,7
        MOV     AL,4CH
        CALL    SENDC
```

```
        MOV     AX,90H
        CMP     SLANT,TRUE
        JE      V1
        MOV     AX,10H
V1:
        ADD     AX,DIR
        CALL    SENDP

        MOV     AX,DC_REG
        OR      AH,64                   ; DGD BIT ON
        CALL    SENDWP
        RET
TEXTE:
        MOV     AL,68H
        CALL    SENDC
        RET

SET_GC:
;
; SET GRAPHIC CHARGER
;
GC1:    IN      AL,0A0H
        TEST    AL,8
        JNZ     GC1

        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     TILE0,0
        MOV     TILE1,0
        MOV     TILE2,0
        MOV     TILE3,0

        TEST    COLOR,1
        JZ      M1
        MOV     TILE0,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     TILE1,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     TILE2,0FFH
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     TILE3,0FFH
```

```
M4:
        MOV     AL,TILE0
        OUT     07EH,AL
        MOV     AL,TILE1
        OUT     07EH,AL
        MOV     AL,TILE2
        OUT     07EH,AL
        MOV     AL,TILE3
        OUT     07EH,AL
        RET

;
; コマンド送出ルーチン
;
SENDC:
        MOV     BX,AX
SENDCLP:        IN      AL,STATPT
        AND     AL,2
        JNZ     SENDCLP
        MOV     AX,BX
        OUT     COUTPT,AL
        RET

;
; パラメータ送出ルーチン
;
SENDP:
        MOV     BX,AX
SENDPLP:        IN      AL,STATPT
        AND     AL,2
        JNZ     SENDPLP

        MOV     AX,BX
        OUT     POUTPT,AL
        RET
;
; パラメータ送出ルーチン(ワード)
;
SENDWP:
        MOV     CX,AX
        CALL    SENDP
        MOV     AL,CH
        CALL    SENDP
        RET

MGPUTC2 ENDP
_TEXT   ENDS
        END
```

11 1ドット描画

CPUで1ドット描画を行う関数mpset()を，リスト11に示します。GDCで行うよりもはるかに高速です。

◉リスト 11

```
;
; 1ドット描画を行なう関数(アセンブリ言語定義)
;
;
; void mpset( x , y , color )
; int x;          /*  点のX座標  */
; int y;          /*  点のY座標  */
; int color;      /*  色( 0≦color≦15)  */
;
;  グラフィックス文字描画
;
;        void mgputc(ascii_code, x, y, color);
;
;        スモールモデル
;        GDC  5MHz
;        グラフィックチャージャ使用
;


;
; スタック上の引数を表わす
; 構造体
;
ARGS     STRUC
ARG_X             DW      ?
ARG_Y             DW      ?
ARG_COLOR         DW      ?
ARGS     ENDS

;
; データセグメント
;
DGROUP   GROUP    _DATA
_DATA    SEGMENT WORD PUBLIC 'DATA'
         ASSUME  DS:DGROUP
```

```
X       DW      0
Y       DW      0
COLOR   DW      0

_DATA   ENDS

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

        PUBLIC  MPSET

MPSET   PROC    NEAR

        PUSH    BP
        MOV     BP,SP

        MOV     AX,[BP+4].ARG_X
        MOV     X,AX
        MOV     AX,[BP+4].ARG_Y
        MOV     Y,AX
        MOV     AX,[BP+4].ARG_COLOR
        MOV     COLOR,AX

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM
;
; SET GRAPHIC CHARGER
;
        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     BX,0
        MOV     CX,0

        TEST    COLOR,1
        JZ      M1
        MOV     BL,0FFH
M1:
        TEST    COLOR,2
        JZ      M2
        MOV     BH,0FFH
M2:
        TEST    COLOR,4
        JZ      M3
        MOV     CL,0FFH
```

```
M3:
        TEST    COLOR,8
        JZ      M4
        MOV     CH,0FFH
M4:
        MOV     AL,BL
        OUT     07EH,AL
        MOV     AL,BH
        OUT     07EH,AL
        MOV     AL,CL
        OUT     07EH,AL
        MOV     AL,CH
        OUT     07EH,AL

        MOV     DI,Y
        MOV     AX,DI
        SHL     AX,1
        SHL     AX,1
        ADD     DI,AX
        SHL     DI,1
        SHL     DI,1
        SHL     DI,1
        SHL     DI,1

        MOV     AX,X
        SHR     AX,1
        SHR     AX,1
        SHR     AX,1
        ADD     DI,AX

        MOV     CX,X
        AND     CX,7
        MOV     AL,80H
        SHR     AL,CL

        MOV     BX,0A800H
        MOV     ES,BX

        MOV     ES:[DI],AL

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET
```

```
MPSET   ENDP
_TEXT   ENDS
        END
```

12 画面フィル

CPUで画面フィルを行う関数mfill( )を，リスト12に示します。これもGDCで行うより高速です。

◉リスト 12

```
;
; 画面フィル(アセンブリ言語定義)
;
;
; void mfill();
;
;       スモールモデル
;       GDC  5MHz
;       グラフィックチャージャ使用
;

;
; プログラムセグメント
;
_TEXT   SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  CS:_TEXT

        PUBLIC  MFILL

MFILL   PROC    NEAR

        PUSH    BP
        MOV     BP,SP

        IRP     REG,<BX,CX,DX,SI,DI,DS,ES>
                PUSH    REG
        ENDM
;
; SET GRAPHIC CHARGER
;
        MOV     AL,0C0H
        OUT     07CH,AL

        MOV     AL,0FFH
        OUT     07EH,AL
```

```
        OUT     07EH,AL
        OUT     07EH,AL
        OUT     07EH,AL

        MOV     AX,0A800H
        MOV     ES,AX
        XOR     DI,DI
        MOV     AX,0FFFFH       ; FILL パターン
        MOV     CX,04000H
        CLD
        REP     STOSW

        IRP     REG,<ES,DS,DI,SI,DX,CX,BX>
        POP     REG
        ENDM

        MOV     SP,BP
        POP     BP
        RET

MFILL   ENDP
_TEXT   ENDS
        END
```

13 画面セーブ

描画例で示した画面セーブを，GDC経由でなくCPUで行う関数gsave2()をリスト13に示します。GDCで行うよりも高速です(これはLargeモデル用)。

●リスト13

```
/*
	グラフィックスVRAMをセーブする
	関数(CPUによる)

	ラージモデル
*/
#include <fcntl.h>

void gsave2(fname)
char *fname;	/* セーブするファイル名 */
{
	int fh;

	fh = open( fname , O_CREAT | O_RDWR | O_RAW , S_IWRITE | S_IREAD);
	write(fh,(char *) 0xa8000L,80*400);
	write(fh,(char *) 0xb0000L,80*400);
	write(fh,(char *) 0xb8000L,80*400);
	write(fh,(char *) 0xe0000L,80*400);
	close( fh );
}
```

14 画面ロード

先に描画例で示した画面ロード関数gload()を,CPUで行うようにした関数gload2()をリスト14に示します(これはLargeモデル用)。

◉リスト14

```
/*
        VRAMにデータをロードする
        関数(CPUによる)

        ラージモデル
*/
#include <fcntl.h>

void gload2(fname)
char *fname;    /* ロードするファイル名 */
{
        int fh;
        fh = open( fname , O_CREAT | O_RDWR | O_RAW , S_IWRITE |
                                                        S_IREAD);
        read(fh,(char *) 0xa8000L,80*400);
        read(fh,(char *) 0xb0000L,80*400);
        read(fh,(char *) 0xb8000L,80*400);
        read(fh,(char *) 0xe0000L,80*400);
        close( fh );
}
```

索引

INDEX

索引

●な行

●は行

●ま行

●ら, わ行

アルファベット順

●A

●B

●C

【参考文献】

『8086 アセンブリ言語』 西村義孝著　日本ソフトバンク刊
『8086 マクロプログラミング』 西村義孝著　日本ソフトバンク刊
『μPD7220 A GDC ユーザーズ・マニュアル(IEM-863D)』
NEC 半導体応用技術本部編　NEC 刊
『Lattice C Compiler Ver.4.1 Reference Manual』
Lattice,Inc., ㈱ LIFEBOAT 著　㈱ LIFEBOAT 翻訳　㈱ LIFEBOAT 刊
『PC-9800 シリーズ テクニカルデータブック』
アスキー出版局 テクライト編　アスキー刊
『PC-9800 シリーズ　テクニカルデータブック　増補改訂 2 版』
アスキー出版局 テクライト編　アスキー刊

GDC テクニカルブック

定価はカバーに記載されております

1989 年 12 月 4 日　第 1 刷発行
1991 年 4 月 8 日　第 4 刷発行
著　者　星野進二
発行者　孫　正義
発行所　ソフトバンク株式会社 出版事業部
〒108　東京都港区高輪 2-19-13
NS高輪ビル
営業☎ 03 (5488) 1360　編集☎ 03 (5488) 1326
印　刷　有限会社バリエ社

Printed in Japan　ISBN 4-89052-090-2
落丁，乱丁本はお取り替えいたします

本書のプログラムは，株式会社 LIFEBOAT の『Lattice C Compiler Ver.4.1』用に書かれたものです。

SOFT BANK ソフトバンク

ISBN4-89052-090-2 C0055 P3500E

定価3,500円(本体3,398円)